Títol: Regulador Solar per a panells de baixa potència

Volum: 1/1
Alumne: Jordi Francès Matas

Director/Ponent: Pere Marés Martí
Departament: ESAII
Data: 18 de Gener de 2011

DADES DEL PROJECTE

Títol del Projecte:Regulador Solar per a panells de baixa potència

Nom de l'estudiant: Jordi Francès Matas
Titulació:Enginyeria Tècnica en Informàtica de Sistemes
Crèdits: 22,5
Director/Ponent:Pere Marés Martí
Departament:ESAII

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President:Juan Climent Vilaró

Vocal:Maria Pilar Muñoz Garcia

Secretari:Pere Marés Martí

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:

Factultat d'Informàtica de Barcelona - UPC

Regulador solar per a panells de baixa potència

Projecte de final de carrera

Jordi Francès Matas
09/01/2011
Enginyeria Tècnica en Informàtica de Sistemes

Índex

1. INTRODUCCIÓ

1.1. SITUACIÓ INICIAL

El Sol és la principal font d'energia del planeta, a més a més és renovable i neta, però no sempre ens és fàcil d'aprofitar per al nostre ús. Pràcticament tota la nostre tecnologia funciona amb electricitat (exceptuant el notori camp dels transports que encara ara és basa, en gran mesura, en la combustió), per tant convertir la radiació solar en electricitat ens és de gran utilitat.

Ja fa anys que es disposa de tecnologies, basades en l'*efecte fotovoltaic,* que permeten realitzar aquesta conversió. Aquest procés però presenta alguns inconvenients:

Les instal·lacions fotovoltaiques no tenen encara rendiments gaire elevats, si més no, no prou elevats com per deixar de dependre d'altres tipus d'energia.

La producció elèctrica està, lògicament, lligada a la irradiació solar en un moment donat. Per tant, tenim problemes de fluctuació en la producció (dies ennuvolats) o de producció nul·la (nits).

Per a contrarestar el primer inconvenient caldria millorar les actuals *cel·les fotovoltaiques* per a que tinguessin rendiments mes elevats. La segona problemàtica té una solució mes fàcil: acumular l'energia en bateries quan es pot produir per a poder-la utilitzar quan no se'n pot produir.

IL·LUSTRACIÓ 1.1-1

L'ús de bateries, al seu torn, presenta un nou problema

Per a poder acumular prou energia per a mantenir, per exemple, la il·luminació durant una nit ens caldran bateries de capacitat mitjana o gran. Aquestes bateries presenten un *llindar de càrrega* alt. El llindar de càrrega és una característica de les bateries, quan es carrega una bateria cal aplicar una intensitat mínima sinó la càrrega no es produeix. A més capacitat té la bateria més alt és aquest llindar de manera que ens caldran panells solars potents per a vèncer el llindar i carrega-les. Aquests panells solars són grossos i costosos.

Aquest motiu dificulta que l'energia solar sigui accessible per a molta gent. No tothom disposa dels diners o de l'espai necessaris per a una instal·lació capaç de carregar bateries mitjanes o grans.

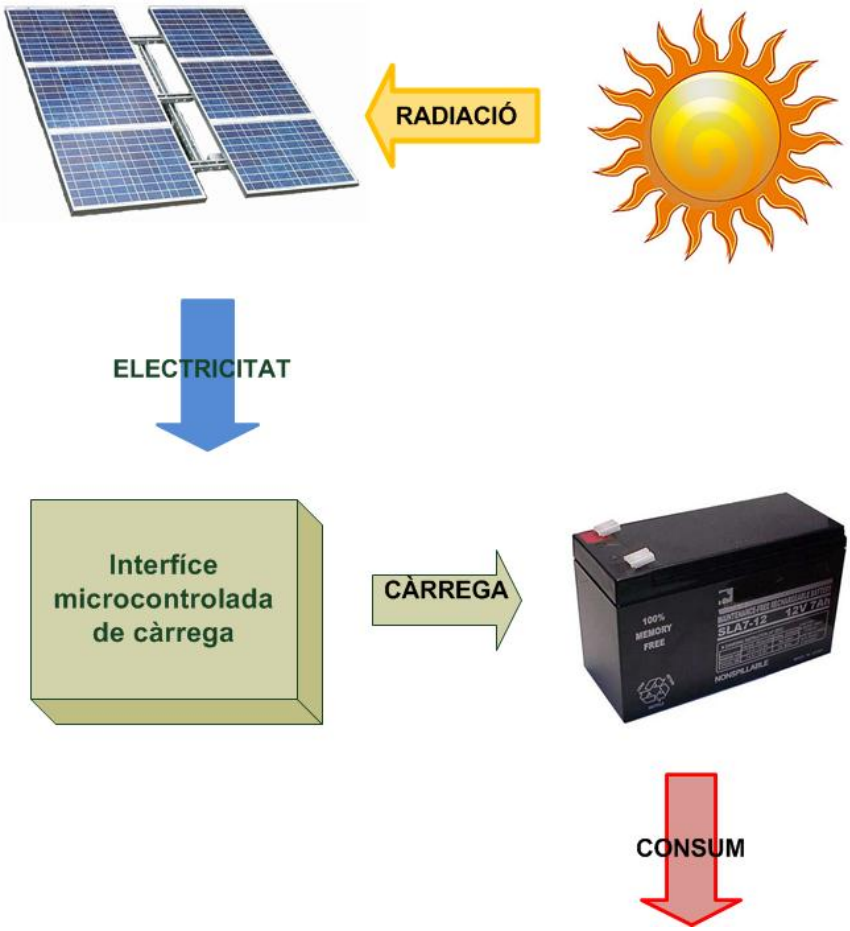
Aquest projecte intenta aportar una solució tècnica a aquest problema tot permetent carregar bateries mitjanes o grans amb panells solars de poca potència, cost moderat i reduïdes dimensions.

1.2. SOLUCIÓ PROPOSADA

Sabem que connectant directament un panell solar petit a la bateria no aconseguirem vèncer el llindar de càrrega. Cal acumular entre la cel·la i la bateria l'energia elèctrica i passar-la cap a la bateria quan hi ha garanties de que pot vèncer el llindar, és a dir, que s'aprofitarà per a la càrrega.

Per tant es dissenyarà i construirà una interfície de càrrega que permeti aquesta acumulació d'energia per garantir la càrrega de la bateria. Aquesta interfície serà governada per un microcontrolador que prendrà les totes les decisions de forma autònoma.

La solució ha de ser autocontinguda, autònoma, barata i de reduïdes dimensions. El compliment d'aquests requisits ens assegura una solució fàcilment desplegable i a l'abast de tothom.



IL·LUSTRACIÓ 1.2-1

1.3. TREBALL PREVI

Aquest projecte es basa en el PFC titulat "Sistema per la gestió de fonts d'energies renovables" d'en Miquel Mariño Espinosa.

En el seu projecte ja es van tractar les qüestions teòriques i es van fer simulacions demostrant que la viabilitat d'una solució. Per tant aquest projecte tindrà un caire més pràctic en abordar la qüestió centrant-se en l' implementació d'una resposta tecnològica al problema.

Tot i que va fer molt bona feina el sistema resultant del seu projecte requeria una font d'alimentació independent de la bateria i el panell, això contradiu la idea d'autonomia que vull aconseguir i és un dels motius més importants per a dur a terme aquest projecte.

1.4. CAMPS D'APLICACIÓ

Els possibles camps d'aplicació d'aquest projecte són molt diversos, de fet qualsevol aplicació que funcioni amb bateria se'n pot beneficiar. Tot i que, lògicament, és especialment útil on es disposi de moltes hores de sol.

Els seus principis de disseny:

- Baix cost
- Mínima intervenció de l'usuari
- Solució autocontinguda
- Reduïdes dimensions

La converteixen en una solució idònia per a mantenir, per exemple, la il·luminació artificial en habitatges a països en vies de desenvolupament, on els habitants no tenen accés a la xarxa elèctrica convencional perquè o bé és inexistent o no se'l poden permetre. Tot i que ens pot semblar insignificant mantenir una il·luminació, basada en LED durant algunes hores un cop es pon el Sol, és en realitat molt important ja que s'allarga la jornada útil.

També seria útil per a instal·lacions autònomes (estacions i boies meteorològiques, parquímetres, repetidors...). Algunes d'aquestes instal·lacions tenen un panell que els recarrega les bateries que fan servir per a funcionar i amb aquesta tecnologia podrien fer servir panells més petits i per tant reduir despeses. D'altres instal·lacions tenen una bateria que cal ser substituïda per una de carregada quan s'exhaureix la càrrega, aquest projecte també podria beneficiar-les permetent-los recarregar-se sense intervenció humana.

1.5. OBJECTIUS

Aquests són els objectius que s'espera assolir amb aquest projecte

1.5.1. GENERALS

• Desenvolupar el maquinari necessari per a connectar una placa fotovoltaica, una bateria i un microcontrolador.

• Desenvolupar el programari necessari per a controlar el procés de càrrega.

• Adequar el programari i maquinari per a mantenir el mínim consum possible i maximitzar la càrrega de la bateria.

• Demostrar la viabilitat d'una solució autònoma.

1.5.2. ESPECÍFICS

• Disseny i implementació d'un prototip funcional del dispositiu.

• Implementació de programari de diagnòstic que permeti verificar el funcionament correcte de les diferents parts del sistema.

• Obtenció de mesures de càrrega, per a garantir el funcionament correcte del sistema.

1.5.3. ACADÈMICS

• Adquirir els coneixements per a muntar un prototip (soldar i muntar un PCB).

• Treballar amb diversos perifèrics d'un microcontrolador (ADC, DAC, FVR,WDT...).

• Crear interfícies entre un microcontrolador i el món físic per poder interaccionar-hi

• Familiaritzar-se amb tecnologies d'estalvi d'energia d'un microcontrolador.

2. MAQUINARI

2.1. CONCEPTE BÀSIC DE FUNCIONAMENT

Com es menciona a la introducció cal desenvolupar maquinari per a permetre una interconnexió controlada entre la bateria i el panell solar. Ja que si es connecta directament el panell a la bateria no es venç el llindar i per tant no es carrega. Cal emmagatzemar l'energia obtinguda del panell per passar-la de cop a la bateria, i anar repetint aquest cicle. Aquest funcionament requereix que la connexió bateria-acumuladors/panell sigui intermitent, de manera que sols permetrem passar el corrent cap a la bateria quan tenim garantia de que s'aprofitarà. Si la connexió fos permanent tota l'energia provinent del panell seria malgastada intentant fútilment vèncer el llindar de càrrega. És important entendre que tot i que la connexió entre el l'acumulador i la bateria és intermitent, el microcontrolador **sempre s'alimenta de la bateria**, independentment de l'estat del transistor, aquesta alimentació independent garanteix la autonomia del sistema.

El sistema permetrà:

- La connexió microcontrolada entre la bateria i el panell.
- Informar al usuari de l'estat del sistema.
- Adaptar-se a canvis de intensitat lumínica

Podríem fer-nos una idea general de tot el sistema amb la il·lustració 2.1-1.

IL·LUSTRACIÓ 2.1-1

Un cicle de funcionament consisteix en acumular prou energia solar convertida en elèctrica per el panell a l'acumulador. El microcontrolador s'adona de que ja tenim l'energia i actuarà sobre la interfície per a que l'acumulador es descarregui cap a la bateria. El microcontrolador monitoritza la descarrega i tornar a separar el l'acumulador de la bateria tant bon punt s'hagi acabat de buidar-lo, així s'inicia el proper cicle. Segons el projecte d'en Miquel Mariño Espinosa aquests cicles durant menys d'un segon. Per tant el sistema esta contínuament executant aquests cicles i a més durant tot el procés si l'usuari així ho desitja pot demanar al sistema informació de la càrrega.

Per a poder complir amb totes aquestes funcions, el hardware disposa de 2 blocs fonamentals: Interfície de càrrega i comunicació amb l'usuari. Aquests blocs lògicament interactuen entre si per oferir totes les funcionalitats.

2.2. TRIA DELS COMPONENTS

Aquests són els components principals que s'han fet servir, i una breu explicació dels motius per els que s'ha triat aquest model en concret. S'han obviat components simples com resistències i condensadors, ja que independentment del fabricant una resistència d'un valor òhmic concret es comporta sempre igual. La mateixa simplificació es pot aplicar als condensadors.

2.2.1. MICROCONTROLADOR

Es el "cervell" del sistema, pren constantment mesures de l'estat del sistema i pren les decisions sobre com cal actuar a cada moment.

S'ha optat per fer servir productes de **Microchip®**. La opció d'un producte d'aquest fabricant fou motivada perquè són els que es fan servir a les assignatures de Sistemes Digitals i Microcontroladors (SDMI) i *Computer Interfaces* (CI) i per tant ja estava familiaritzat amb l'arquitectura i les eines de desenvolupament.

El model concret: **PIC16F1827**, s'ha triat d'entre els centenars de models del fabricant perquè es trobava a la intersecció entre la potència de càlcul necessària, el baix consum, els pins d'entrada/sortida i el perifèrics necessaris.

Per el que fa a la potència de càlcul, la presa de decisions que ha dur a terme el processador no és computacionalment gaire complexa i els perifèrics necessaris (USART,ADC...) són comuns a pràcticament tots els microcontroladors per tant una MCU de la família **PIC16** es suficient per a aquest projecte. Cal tenir en compte que a més potència més car i més

consum i per tant ajustar el dimensionat del microcontrolador és important per a complir els objectius.

Mantenir un consum baix és imprescindible, si la gestió de la càrrega, consumeix massa energia tindríem càrregues lentes fins i tot descàrregues. Microchip® fabrica alguns dels seus models amb tecnologia **XLP®** (*eXtreme Low Power*), és a dir tecnologia de molt baix consum. Aquests xips tenen consums especialment baixos mentre dormen, i per software garantirem que el microcontrolador es mantingui adormit sempre que sigui possible.

Dormir, per a un microcontrolador, és un estat de funcionament anàleg a la hivernació dels animals, és a dir redueixen el seu funcionament i consum al mínim (~30nA). De fet, és un estat de funcionament alterat, ja que la MCU, no segueix amb el flux d'execució normal (apaga els rellotges i per tant li resulta impossible fer-ho). Alguns dels perifèrics si que romanen en funcionament, per exemple l'ADC, i poden produir interrupcions que "desperten" el micro, el tornen a el seu funcionament habitual. Els conceptes de dormir i interrupció es tracten més detalladament al capítol de programari.

Per el que fa a l'entrada/sortida aquest PIC disposa de 13 pins d'entrada sortida (5 al port A + 8 al port B), necessaris per fer interactuar el micro amb la resta del sistema. El nombre de pins d'entrada sortida va ser fonamental per a l'elecció del microcontrolador ja que n'hi havia d'altres com el PIC16F1823 que complien amb la resta de requeriments però els faltaven capacitats d'entrada/sortida.

Aquesta MCU pertany a la subfamília PIC16F18xx, que està basada en la PIC16 però té característiques de la PIC18. Podríem pensar-hi com un híbrid entre PIC16 i PIC18. Entre les característiques "esteses" hi trobem:

- Pila de crides de 16 nivells.
- Registres “ombra” per a les interrupcions, el context es guarda per hardware.
- Registres FSR per a l’adreçament.
- Suport per a freqüències de rellotge de fins a 32 Mhz.
- Joc estès d’instruccions
- WDT estès

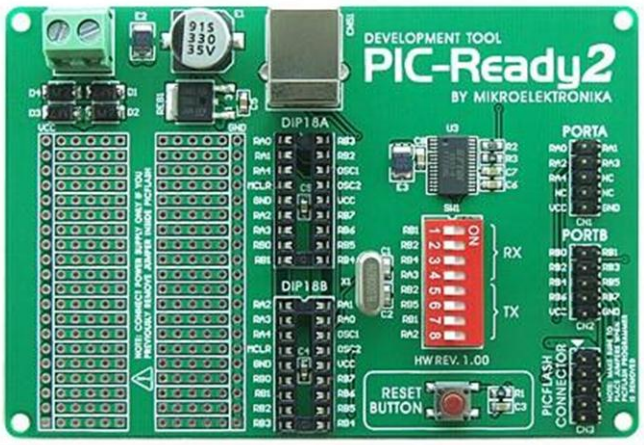
Aquestes característiques, tot i que no van ser cabdals en la decisió, s’han aprofitat en aquest projecte, l’ús que se’n ha fet està detallat a l’apartat de software.

2.2.2. PLACA DE DESENVOLUPAMENT

És el suport per al microcontrolador, conté reguladors de tensió i corrent per a l’alimentació del microcontrolador. Hi ha plaques amb molt de hardware addicional (potenciòmetres, polsadors, pantalles LCD...) com per exemple els *trainers* que es fan servir a SDMI o CI. Per a aquest projecte, intentar seguir una línia minimalista, s’ha agafat la mínima expressió d’una placa de desenvolupament.

Concretant, s’ha triat una PIC-Ready2® de Mikroelektronika® que és exactament el que cal per al projecte. Ofereix un 2 sòcols PDIP-18 (un d’ells compatible amb el PIC16F1827) i interfícies **IDC10** per a tenir fàcil accés als pins d’entrada/sortida.

L’únic hardware, no essencial, que ofereix aquesta placa és un xip ***FTDI*** que permet que els ordinadors puguin interactuar-hi detectant-la com un port sèrie (COM), simplificant molt el software del PIC (ja no cal gestionar transferències USB) i afegint connectivitat USB a microcontroladors que, com en el nostre cas, no en tenen.



IL·LUSTRACIÓ 2.2-1

2.2.3. TRANSISTOR IRF-740

Per a poder interrompre o deixar circular el corrent des de l'acumulador intermig (condensadors d'alta capacitat) cal algun dispositiu que actuï com a interruptor.

Ens em decantat per aquest model ja que és MOSFET i per tant té un consum molt baix en mode estàtic. Concretament un IRF-740, perquè admet corrent de pols molt alta (~40 A) i és ideal per a les descàrregues del condensador cap a la bateria. A mes presenta una resistència interna pràcticament negligible (< 0.55 Ω) en saturació.

2.2.4. OPTOACOBLADOR TIL-111

L'IRF740 té una capacitat paràsita a la gate de 1400 pF i per tant no la podem activar directament des del PIC, ja que amb la intensitat que el micro no n'hi ha prou per aconseguir commutacions ràpides. Cal emprar un dispositiu que amplifiqui la corrent del micro per a poder dirigir el transistor. Tot i que podríem haver optat per un transistor més petit i que si que pogués ser disparat per el micro. Es va optar per un optoacoblador perquè a més de permetre'ns governar còmodament l'IRF740 també ens aïlla òpticament la lògica (microcontrolador) de l'etapa de potència (transistor).

2.3. INTERFÍCIE DE CÀRREGA

És la part principal del projecte, permet connectar la bateria i el panell solar entre ells i també permet al microcontrolador llegir l'estat del sistema per prendre les decisions sobre quan i com actuar sobre el circuit.

ESQUEMA 2.3-1

2.3.1. ESTRATÈGIES DE COMMUTACIÓ

Com s'ha vist anteriorment, la connexió entre bateria i panell ha de poder ser interrompuda de forma controlada, lògicament farem servir un transistor per a controlar la connexió amb el microcontrolador, concretament l'IRF740. Però hi ha dues possibilitats segons on s'instal·li el transistor: o bé tallar la massa o bé tallar el positiu. Ambdues opcions ofereixen avantatges i inconvenients per tant s'han considerat totes dues en diversos prototips.

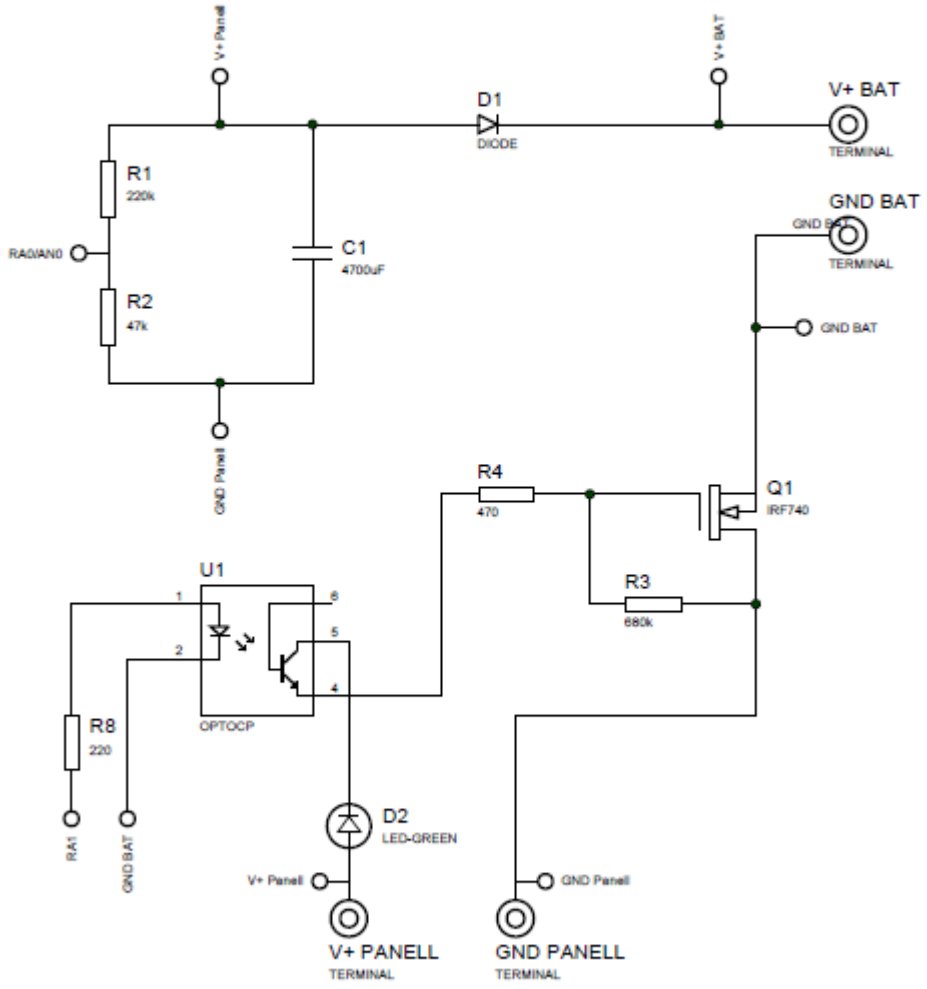
TALLAR LA MASSA

Tot i que l'habitual per tallar un circuit és tallar-lo per el positiu, en el nostre cas: instal·lar el transistor entre V+ Panell/V+ Condensador i V+ Bateria. També és possible fer aquest tall per el costat de les masses, es a dir muntar el transistor entre GND Bateria i GND panell. Optar per aquesta segona opció ens permet mantenir un maquinari més simple ja que no requereix treballar amb tensions altres.

Es va muntar un prototip basat en la idea de tallar la massa.

PRIMER PROTOTIP

És un circuit força senzill, com es pot veure en l'esquema 2.3-1.

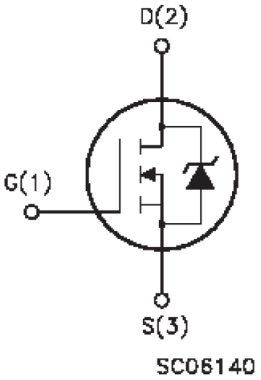


ESQUEMA 2.3-2 PROTOTIP 1

El funcionament general és el descrit a l'inici d'aquest capítol. Particularitzant a aquest model veiem que l'energia elèctrica provinent dels panells s'acumula en el condensador C1. Un cop ple el microcontrolador dona l'ordre a traves del seu pin RA1 a l'optoacoblador que fa circular el corrent i satura el transistor IRF740 tot unint les masses i permetent la

circulació de corrent del condensador cap a la bateria. La resistència R3 és necessària com a *pull down*, garanteix que la capacitat parasita de la gate del MOSFET es pugui descarregar amb l'optoacoblador tallat. És fàcil veure sobre l'esquema 2.3-2 que sense aquesta resistència no hi hauria escapament per al corrent, que deixaria sempre satura el transistor i per tant impediria el funcionament de la interfície de càrrega.

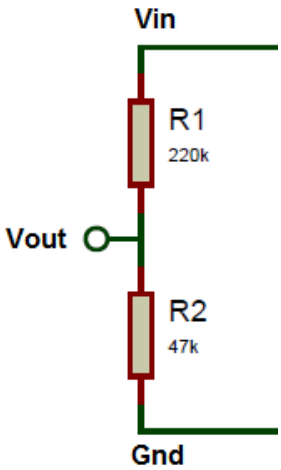
El díode D1, que apareix a l'esquema es necessari ja que a l'esquema del transistor (esquema 2.3-4) podem veure que hi ha un **díode Zener** entre sortidor i drenador, que podria conduir un corrent indesitjable, D1 evita la circulació d'aquest corrent.



ESQUEMA 2.3-4

Tot i que en l'esquema apareix un sol condensador C1 de 4700 µF en el maquinari hi ha 2 tires de pins femelles, permetent connectar diversos condensadors en paral·lel tot sumant-ne les capacitat. Així es poden fer proves empíriques per veure quins condensadors donen millors resultats.

EL DIVISOR DE TENSIÓ:



La seva funció és "canviar l'escala" dels voltatges ja que el microcontrolador només pot llegir voltatges entre 0V i 5V respecte el seu terra .

RESULTATS DE TALLAR LA MASSA:

El prototip aconseguia commutar correctament el transistor connectant i desconectant les masses tot passant càrrega del condensador cap a la bateria.

Presentava un inconvenient notable: no era capaç de llegir correctament l'estat de càrrega del transistor. El PIC està sempre alimentat per la bateria, per tant el seu terra es el de la bateria i en consecuencia només pot mesurar tensions de fins a 5V respecte el terra de la bateria. Amb el plantejament de tall de massa durant gran part del temps el condensador no té la massa en comú amb la bateria, i per tant no és llegible per al PIC. Només es pot fer la lectura correcta quan l'IRF740 està saturat i per tant les masses son comunes, però tant bon punt les masses es possen en comú comença la descàrrega del condensador. Aixó ens porta a la paradoxa de no poder fer lectures acurades ja que els intents de lectura malmeten la magnitud a llegir.

Si no és capaç de llegir correctament les tensions és impossible que pugui pendre les decisions de quan cal conmutar el transistor correctament. Per tant es va intentar abordar el problema per l'altre vesant: tallar el positiu.

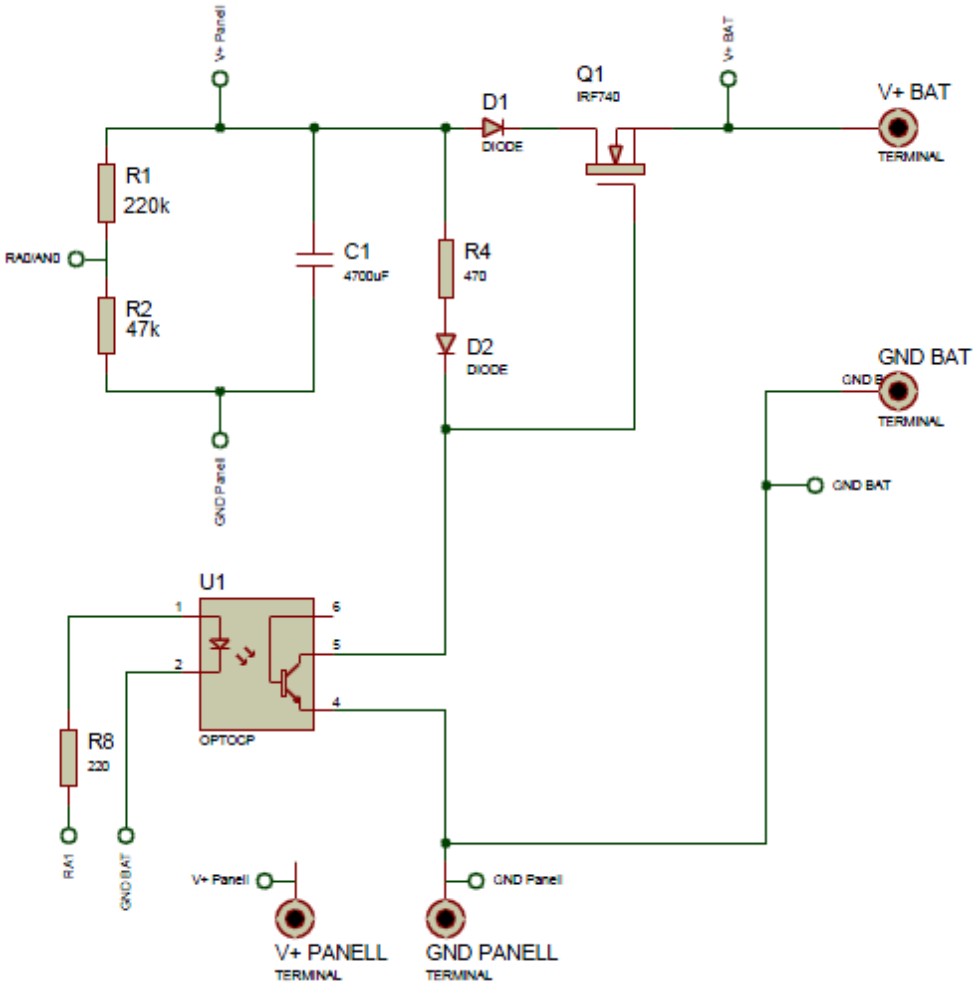
TALLAR EL POSITIU

Naturalment aquesta opció consisteix en situar el transistor IRF740 a la branca del positiu per a poder-la interrompre a voluntat. Cal tenir en compte que per els requeriments de voltatge que té aquest mètode també s'incrementen lleument els requeriments de components.

Es va construir un prototip per estudiar la possibilitat del tall de positiu.

SEGON PROTOTIP

Podem veure l'esquema 2.3-5 el prototip de tall de massa:



ESQUEMA 2.3-5 PROTOTIP 2

Tot i que el funcionament bàsic es el mateix que en el model anterior: interrompre la connexió entre el condensador i la bateria de forma controlade per aconseguir la càrrega. En aquest es talla per la branca positiva i hi ha algunes diferencies significatives en el connexionat per aconseguir els voltatges necessaris per a controlar el transistor. En aquesta versió el transistor està per defecte saturat ja que rep el corrent directament del panell i el micro pot descarregar-ne la gate donant una ordre que fa obrir el camí de descàrrega a l'optoacoblador. El díode D2 és necessari per evitar descàrregues involuntàries del transistor.

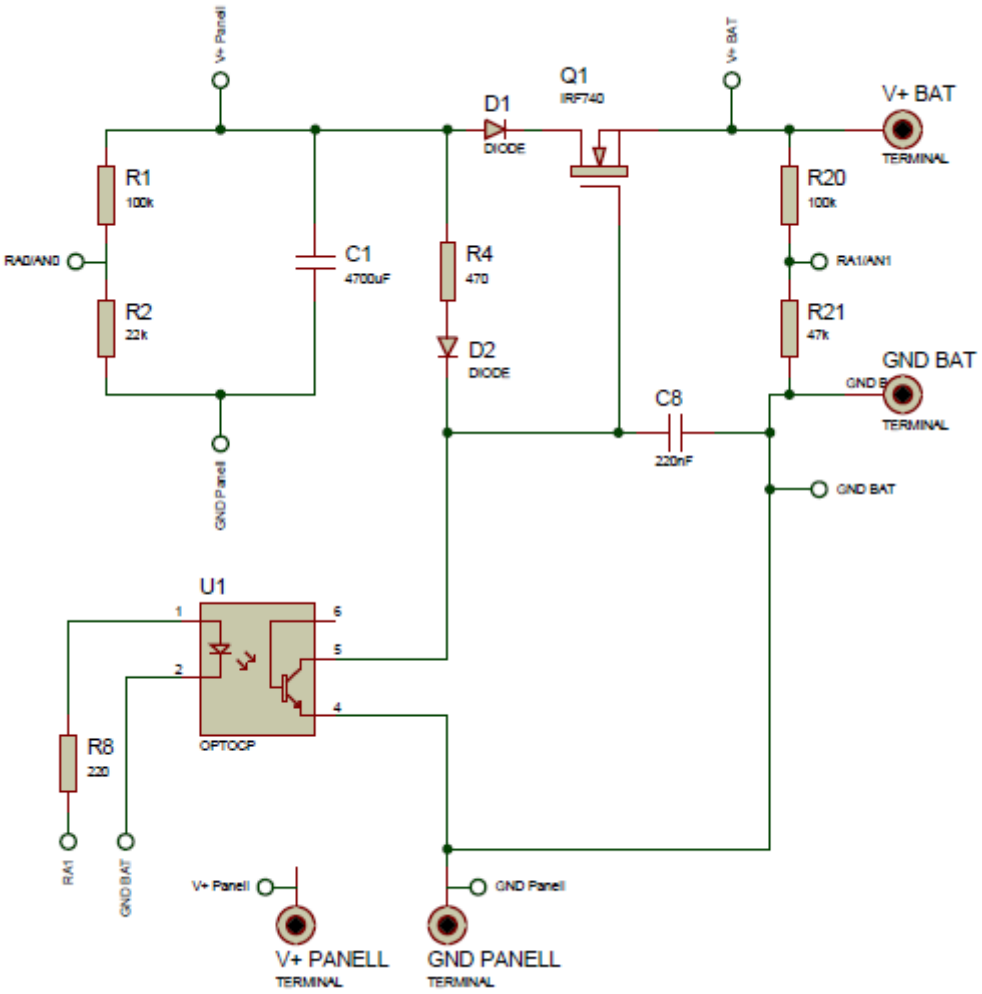
Tenint en compte la diferencia de “estat per defecte” va caldre modificar lleugerament el programari (negar la senyal que controlava l’optoacoblador) per a fer funcionar el model.

RESULTATS DE TALLAR EL POSITIU:

La millora més notable respecte el model anterior es que ara es pot llegir sempre la tensió del condensador ja que les masses sempre estan en comú i per tant les referencies son vàlides per a fer la mesura. Tot i que el model encara no es perfecte, ja que quan satura el condensador oliga a la branca del panell a baixar a el voltatge a que està la bateria i en baixar el panell es perd la tensió necessària per a la saturació completa i el transistor entra en zona activa tot oposant resistència entre el condensador i fa aparèixer una diferencia de potencial entre les dues branques.

2.3.2. VERSIÓ FINAL

Tenint en compte que tallar el positiu va donar més bons resultats que no pas l'alternativa s'ha optat per aquest model per a la versió final tot afegint-hi petites modificacions per pal·liar els problemes del prototip de tall de positiu.



ESQUEMA 2.3-6 VERSIÓ FINAL

Es pot veure a l'esquema 2.3-6 que s'han afegit alguns components a aquesta versió per pal·liar els problemes de l'anterior. El problema de la diferencia de potencial es resol afegint un condensador de 220 nF en

paral·lel a la capacitat parasita de la gate, d'aquesta manera la combinació de la capacitat parasita, el condensador de 220 nF i el díode D2 conformen un detector de pic que no es buida per corrents indesitjables com en el model anterior.

A més a més s'ha afegit un divisor de tensió propi per a la branca de la bateria i s'ha redimensionat el de la branca del condensador, totes dues accions permeten pendre unes mesures mes acurades del estat del sistema.

2.3.3. COMUNICACIÓ AMB L'USUARI

Només amb la interfície de càrrega tindríem un sistema mut, a simple vista no es podria saber si està funcionant adequadament o no, per tant és necessari afegir-hi interfícies de comunicació amb l'usuari.

S'ha optat per una pantalla de caràcters LCD i aprofitar el port sèrie del microcontrolador ja que ambdues solucions són efectives i de cost i complexitat adequats al projecte.

Pot semblar estrany afegir-hi un LCD ja que sembla contradir el principi de simplicitat en el maquinari; però cal assegurar que el disseny és autocontingut i si només féssim servir el port sèrie estem condicionant l'ús a la presencia d'un ordinador capaç de llegir-lo.

2.3.4. PORT SÈRIE

És una interfície de comunicació molt habitual en els sistemes encastats. Molt microcontroladors, com el PIC16F1827, disposen de unitats de maquinari especifiques per a gestionar aquest tipus de port.

Gràcies al FTDI la nostra placa de desenvolupament (PIC-Ready2) podem treballar amb l'UART de la forma habitual i ja es convertirà a USB de forma transparent tot facilitant la connexió amb ordinadors actuals; que cada vegada prescindeixen més d'un port sèrie físic.

L'ús i la configuració del port es detallen a l'apartat de programari.

2.3.5. PANTALLA LCD

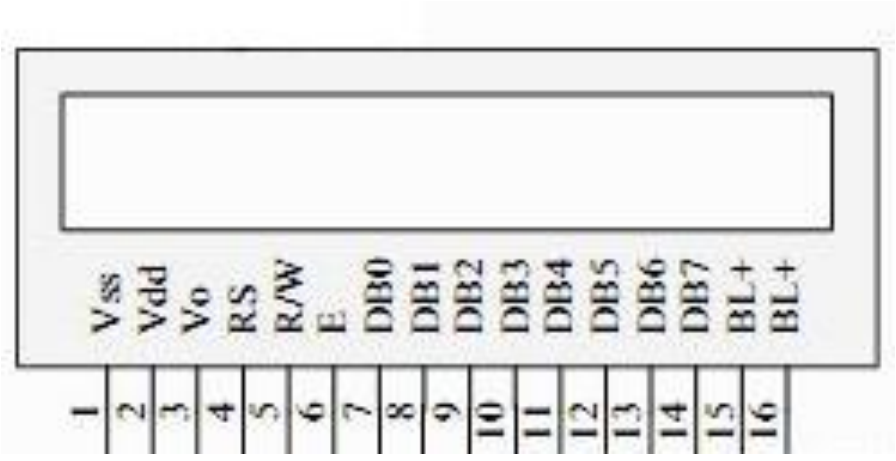
Les pantalles alfanumèriques LCD també són força habituals en els sistemes encastats.

A diferencia d'una pantalla convencional d'ordinador que pot admetre diferents resolucions i no té uns patrons definits, una LCD alfanumèrica té un nombre finit de cel·les i a cada una hi pot representar un dels patrons que té a la ROM. Tot i que el sistema de patrons pugui semblar molt limitat la realitat és que aquests patrons inclouen tot l'alfabet (majúscules i minúscules), els nombres, signes de puntuació i alguns caràcters especials. Fins i tot es poden definir alguns caràcters personalitzats.

Permeten oferir informació de forma rapida i concisa. Són una solució totalment autocontinguda (no cal cap hardware addicional). A més tothom pot llegir una pantalla, de manera que fa el producte útil a gent sense formació tècnica.

2.3.6. INTERFÍCIE DEL LCD

Les pantalles LCD segueixen un estàndard *de-facto* per a la seva connexió amb un sistema. Una tira simple de 16 pins, on els 14 primers són per la alimentació i comunicació amb la lògica de la pantalla i els 2 darrers són l'alimentació de la retroiluminació.

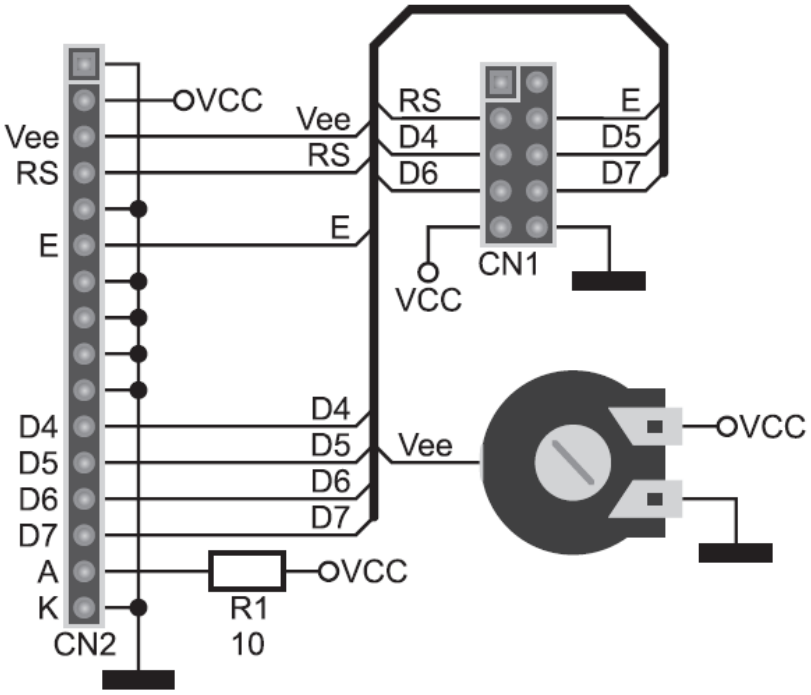


IL·LUSTRACIÓ 2.3-1

FIGURE 2.3-1

Per connectar l'LCD amb la resta del sistema utilitzarem un adaptador de Mikroelektronica que és compatible amb aquest estàndard de 16 pins i amb els connectors IDC-10 de la placa de desenvolupament.

Cal considerar que un LCD amb retroiluminació té un consum elevat (~100 mA), per tant no el podem afegir a un disseny de baix consum sense prendre algunes de control. Per assegurar que el consum del sistema no sigui innecessàriament alt, la pantalla LCD només estarà encesa quan l'usuari ho demani. Caldrà modificar l'adaptador per a poder engegar i parar la pantalla ja que per si sol no ofereix aquesta funcionalitat.

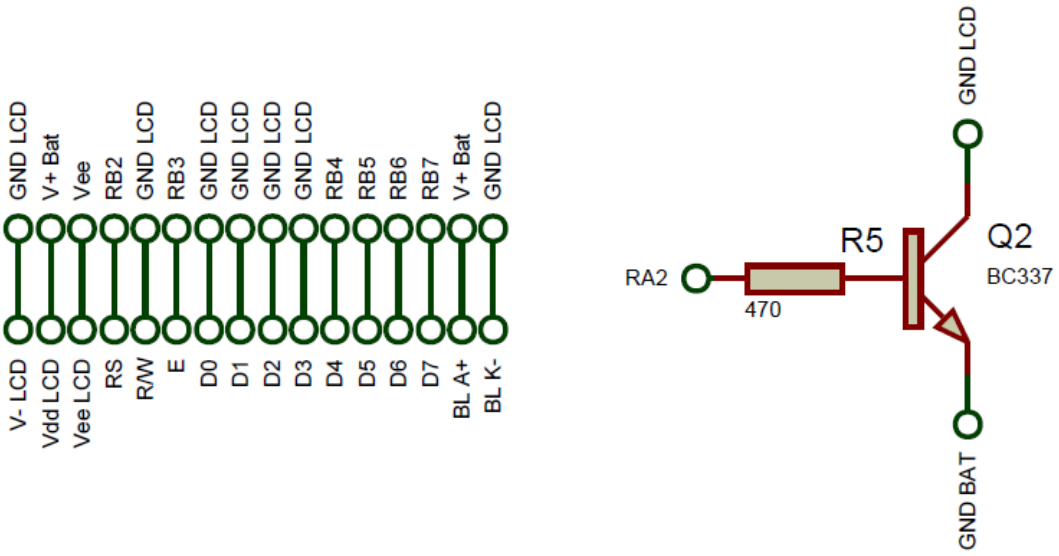


ESQUEMA 2.3-7

Es pot veure en l'esquema 2.3-7 que ens obliga a treballar amb mode de 4 bits, així ens estalviem uns quants pins d'entrada sortida, que podrem dedicar a altres funcions, aquest aspecte s'aborda amb mes profunditat en el capítol de programari.

L'adaptador pren l'alimentació del IDC-10 (CN1) que connecta a la placa de desenvolupament, per tant si interrompem la circulació del corrent o bé a VCC o bé al GND, l'LCD perdrà l'alimentació i per tant s'aturarà. Quan restaurem la circulació del corrent recuperarem el funcionament de l'LCD.

És necessari recórrer novament a un transistor, tot i que aquesta vegada cal gestionar corrents molt més petits i es va optar inicialment per un BC337, que permetia un circuit molt simplista (esquema 2.3-8).



ESQUEMA 2.3-8

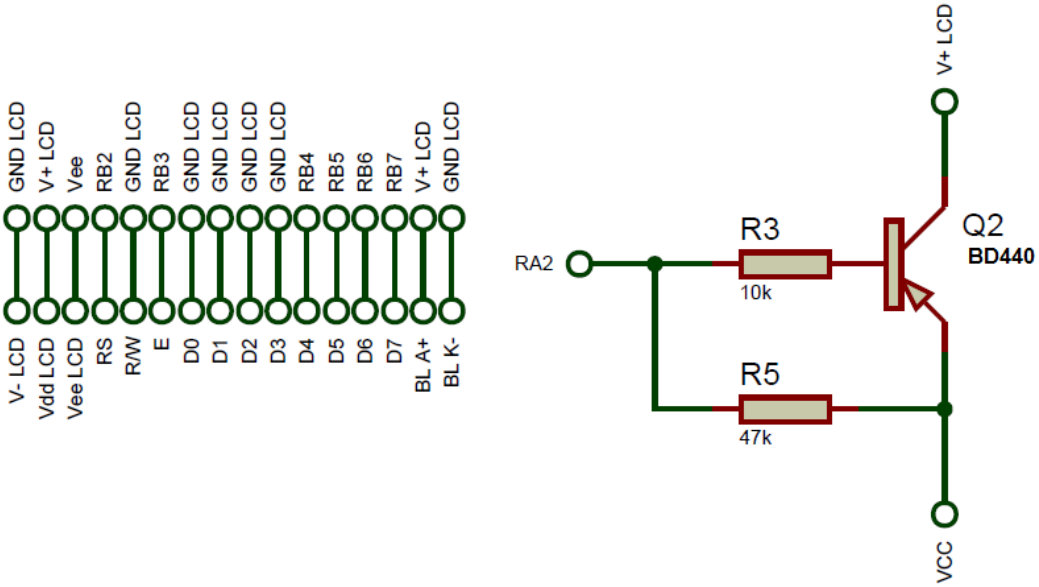
GND LCD que no es connecta directament amb el GND BAT, sinó que passa per el BC337. D'aquesta manera amb una petita corrent a la base del transistor permetrem la circulació del corrent tot encenent tant la lògica com la retroiluminació del LCD. És per tant una aplicació del concepte de tallar la massa.

Un cop muntat el maquinari del LCD i s'ha adaptat el programari per a que el faci servir es van poder observar comportaments adversos. Quan l'LCD estava en marxa, és a dir que permetíem la circulació del corrent a través del transistor , tot funcionava adequadament.

Però quan es tallava el transistor, l'LCD era capaç d'agafar terra dels pins de dades que tingués a *low* (a 0 lògic), i com que no disposava de prou intensitat es quedava en un estat on no funcionava plenament però consumia contínuament.

Per evitar més modificacions al hardware la meva primera aproximació va ser solucionar-ho en el codi. Si en el moment de parar l'LCD deixava tots els pins relacionats amb l'LCD a 1 no podria agafar terra. Aquesta solució però implicava mantenir el bit de *ENABLE* a 1, de manera que qualsevol soroll a les pistes, o per exemple l'USART (un dels pins és multiplexat) produïen efectes col·laterals en la pantalla. Tots aquests inconvenients van forçar a prendre un nou enfocament.

La següent opció va ser, com en la interfície de càrrega, tallar el positiu, va implicar canviar el transistor per un de característiques similars però PNP seguint l'esquema 2.3-9.



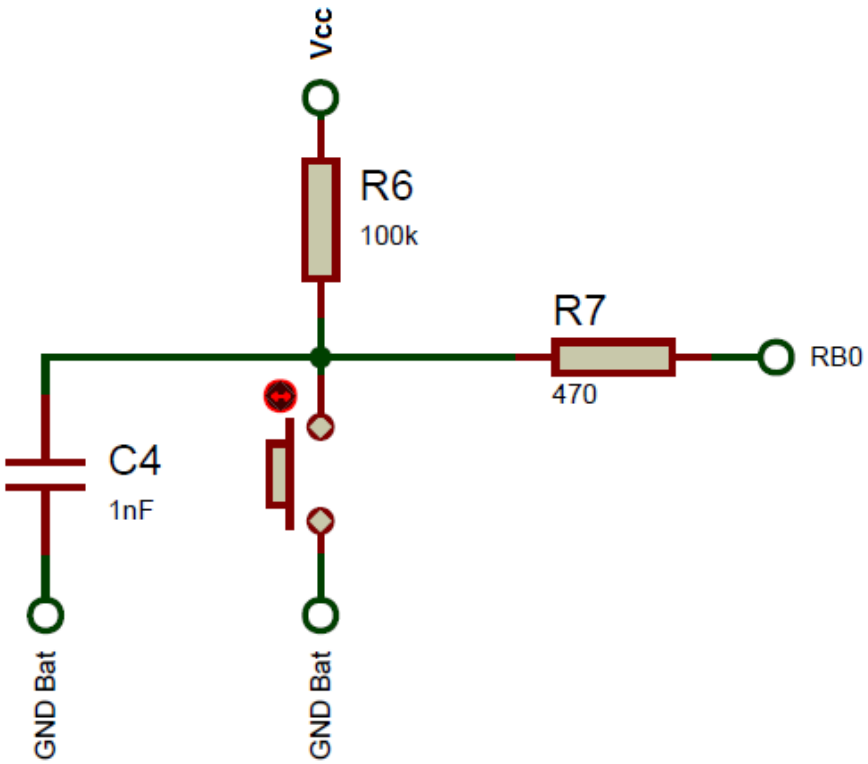
ESQUEMA 2.3-9

Tot i els canvis en el maquinari i el tipus de transistor el funcionament, a nivell conceptual, es manté. El canvi implica afegir una resistència de pull up per garantir que talla completament l'alimentació a l'LCD. Amb aquest mètode es tallen els 5V, per tant ni que l'adaptador fos capaç d'agafar alimentació de les línies de dades n'hi ha prou amb mantenir-les a 0, evitant així tots els problemes amb el bit d'Enable mencionats anteriorment.

Però ara que ja podem controlar quan volem que l'LCD estigui en funcionament, cal algun mecanisme que permeti a l'usuari indicar que vol rebre informació amb l'LCD.

S'opta per afegir un polsador, és una opció barata i efectiva, a més a més tothom sap prémer un botó. Per tant serà una opció compatible amb la idea de poder desplegar aquesta solució a tot arreu i que no calgui formació prèvia per utilitzar-la.

La instal·lació del polsador es fa seguint l' esquema 2.3-10:



ESQUEMA 2.3-10

L'esquema mostra que el polsador treballa directament amb els voltatges del microcontrolador. S'ha instal·lat un petit condensador (C4) per evitar els rebots.

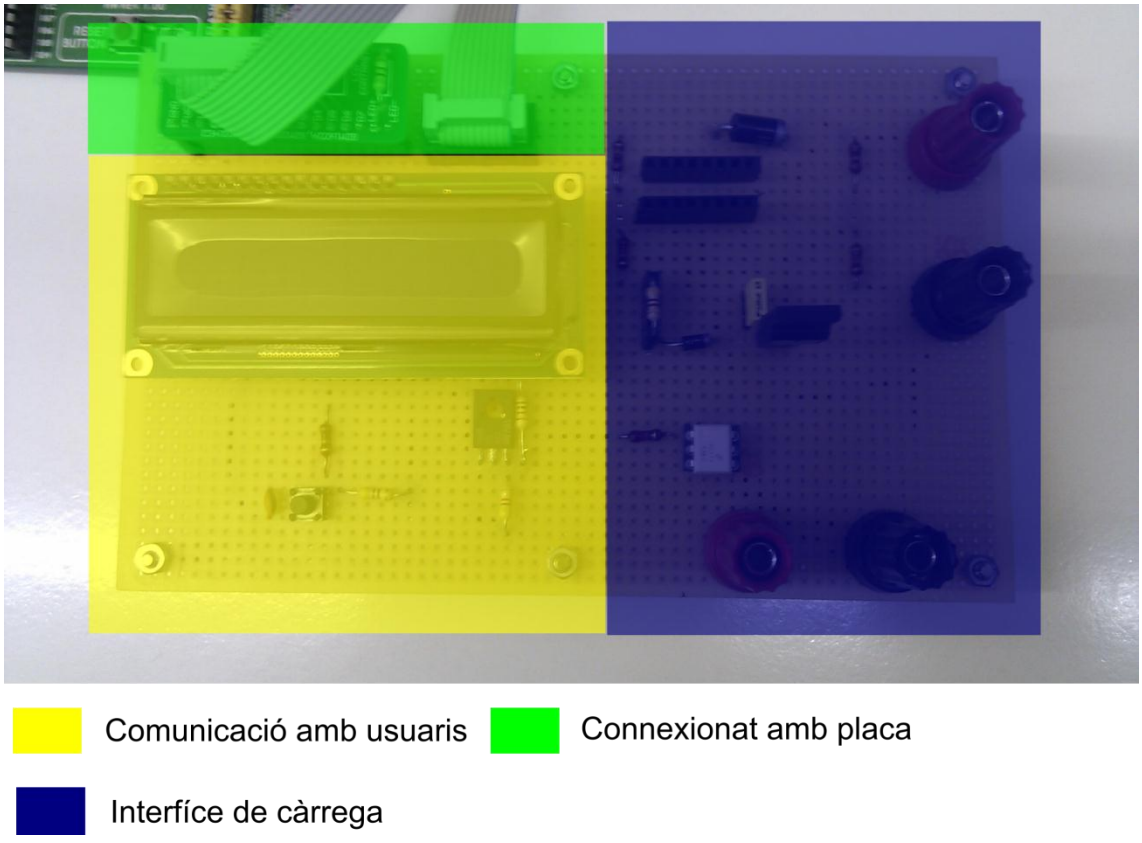
En repòs (quan el polsador no està premut) el condensador es va carregant fins a Vcc (5V), per tant el micro veu 5V (1 lògic) a la seva entrada RB0. En prémer el polsador es descarrega el condensador, que no es tornarà a carregar fins que no es deixi anar el botó i en aquest moment la MCU llegeix 0V (0 lògic). S'ha aprofitat el pin d'interrupció externa del PIC per a la detecció de la pulsació (aquest aspecte està mes detallat en el capítol de programari).

2.4. CONSIDERACIONS PRÀCTIQUES DEL MAQUINARI

Fins ara s'han considerat els aspectes teòrics del maquinari però lògicament totes les versions que s'han mencionat fins ara s'han anat implementant físicament.

2.4.1. COL·LOCACIÓ DELS COMPONENTS

Tot i que pot semblar un detall sense importància la col·locació dels components sobre la placa perforada no és en cap moment aleatòria. S'ha intentat mantenir una divisió de l'espai en àrees funcionals de manera que d'un cop d'ull es pot saber a quin bloc pertany un element.



IL·LUSTRACIÓ 2.4-1

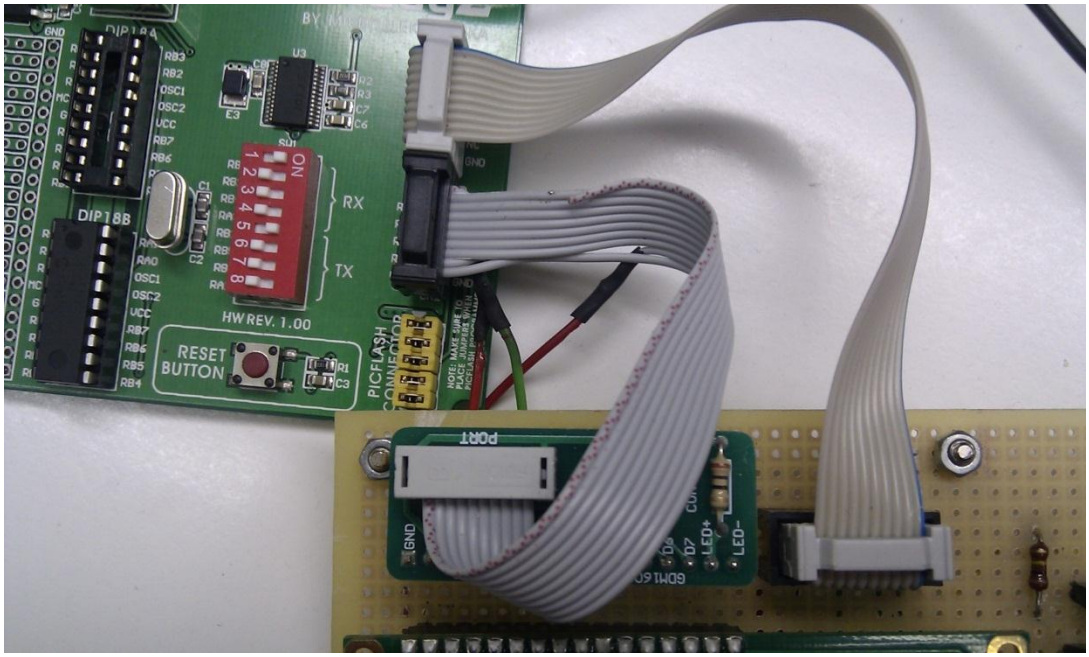
2.4.2. CONNEXIÓ AMB LA PLACA DE DESENVOLUPAMENT

Inicialment es van tirar cables individuals per a connectar cada pin amb el component corresponent però aquest mètode rudimentari fa molt lent el muntatge i indueix a l'error (punxant molts cables és fàcil que un no es connecti al pin correcte). A més a més és molt inestable, petits

moviments de la placa o a vegades la tensió dels mateixos cables els pot fer desconnectar.

Per a evitar aquests problemes es va optar per afegir un connectors 5+5, compatible amb l'IDC10 de Mikroelektronika. Així amb un sol cable pla és possible dur totes les senyals de control del circuit.

A el port B s'hi connecta l'adaptador de l'LCD, modificat a l'alçada del cable pla, on s'intercepten alguns cables. Per tal que l'LCD quedes fixat a la placa perforada i no penjant del cable pla s'ha afegit uns sòcols per a l'LCD i el seu adaptador. La pista del cable pla del port B que es correspon a el bit 0 s'intercepta per a poder-lo connectar al polsador. Cal recordar que RB0 té associada la lògica de la interrupció externa i per tant aquesta funció està sempre vinculada a aquest pin. D'aquesta manera aprofitem el cable de l'LCD per passar una funció més.



IL·LUSTRACIÓ 2.4-2

3. PROGRAMARI

Ara ja tenim el maquinari, però és passiu, no prendrà cap acció a no ser que un programa així li ho indiqui. En aquest capítol es tracta la besant de programació d'aquest projecte.

3.1. ENTORN DE DESENVOLUPAMENT

Hi ha moltes eines per a desenvolupar programes per a diferents arquitectures de computador, s'ha optat per les següents:

3.1.1. MPLAB

És l'entorn de desenvolupament més habitual per a treballar amb productes de Microchip, de fet el proporciona el mateix fabricant de forma gratuïta. Ofereix integrades funcions per a desenvolupar el codi, per a gravar-lo al PIC i per a corregir-lo. S'hi poden instal·lar compiladors de llenguatges d'alt nivell per no limitar el desenvolupament dels nostres programes al ASM de PIC. A més a més ja hi estava familiaritzat. Són aquests els motius que em van fer optar per l'IDE de Microchip.

3.1.2. HI-TECH C COMPILER

Tot i que *l'assambler* permet fer codi molt òptim per a l'arquitectura és difícil fer codis relativament grans en assambler i per això recorrem a llenguatges de més alt nivell. En aquest projecte he optat per el C ja que és força utilitzat per a microcontroladors i ofereix la possibilitat de operar a baix nivell si és necessari. Es van considerar tres compiladors:

- HI-TECH C Compiler
- CCS C Compiler
- mikroC

Tot i que tots ofereixen suport per als dispositius més comuns de Microchip; en el moment de fer la tria només el HI-TECH C Compiler oferia suport per al PIC16F1827 i per tant es va convertir en l'únic candidat viable.

Cal mencionar que la majoria dels compiladors per a microcontroladors ofereixen extensions al C estàndard, per exemple tipus de dades especials per a micros o directives per a treballar amb interrupcions.

3.2. PECULIARITATS DEL DESENVOLUPAMENT PER A PIC

Cal mencionar algunes de les peculiaritats i limitacions forçades per l'arquitectura d'un PIC:

- No hi ha sistema operatiu, per tant no es pot sortir mai del programa cal "atrapar" la execució. Molts compiladors actuals ja creen codi que obliga a tornar al principi del main si s'arriba al final però preferible fer-ho de forma explicita amb estructures del llenguatge.
- No hi ha ALU per a nombres de coma flotant, de forma nativa els PIC no poden treballar amb comes flotants. Molts compiladors ofereixen tipus de dades de coma flotant que de forma transparent a l'usuari són gestionades per software, és una solució molt lenta i també és costosa en memòria ja que per una única operació amb operands amb coma flotant s'afegirà a el codi totes les funcions necessàries per a tractar-los. En el codi del projecte s'ha evitat fer servir nombres amb coma flotant.

Un PIC es basa en l'arquitectura **Harvard** , és a dir la memòria de dades i la d'instruccions estan físicament separades.

Aquesta separació no hi és en els PCs (basats en la arquitectura **von Neumann**), és una distinció important ja que implica que un punter a una variable (emmagatzemada en memòria

de dades) no pot apuntar a una constant (emmagatzemada en memòria de programa) i per tant pot caldre duplicar petites parts de codi per a fer la mateixa operació depenent de l'origen de les dades.

- La memòria de dades és molt limitada (384 bytes) per tant cal minimitzar l'ús de variables. Habitualment n'hi ha prou amb guardar les cadenes llargues de text com a constants (directiva **const** del compilador) ja que s'emmagatzemaran en memòria de programa.
- La pila de crides és de 16 nivells, per tant cal vigilar amb l'aniuament excessiu de crides, cal tenir especial cura si s'utilitzen interrupcions ja que poden ser crides entrants en qualsevol punts de l'execució. En el programa s'ha evitat excedir aquest nivell.
- El mecanisme d'interrupcions permet llançar codi específic per a tractar un esdeveniment que es pot produir en qualsevol moment. Quan succeeix l'esdeveniment diem que s'ha produït una interrupció, i si està permesa l'MCU desviarà l'execució cap a la rutina d'interrupcions (situada a 0x04 de la memòria de programa).
- El mode de baix consum d'un micro és un mode de funcionament anormal, on moltes de les funcions s'aturen i s'aconsegueix un consum baix (~30nA). Normalment ens referim a aquest mode com a adormir el micro, per a sortir d'aquest mode i retornar al funcionament normal cal que es produeixi un esdeveniment(interrupció) que puguin despertar el micro.

3.3. PARTS DEL PROGRAMARI

Es poden diferenciar 2 components del programari desenvolupat per a aquest projecte: un carregador d'arrencada (***bootloader)*** i el programa de control.

3.3.1. BOOTLOADER

Un *bootloader* és un petit programa capaç de rebre el codi d'un programa més gran i gravar-lo a la memòria del microcontrolador. Són molt útils per a accelerar el procés de desenvolupament d'aplicacions per a una MCU ja que ens estalvien haver de treure el PIC a cada canvi que fem al software.

Lògicament un *bootloader* fa ús d'alguna interfície del microcontrolador per rebre el codi del programa. Es poden trobar molts *bootloaders* diferents en funció de quina interfície utilitzen o per a quin dispositiu estan dissenyats.

Sol anar acompanyat d'una aplicació per a ordinador que permetrà fer l'enviament del codi a gravar i executar. Alguns bootloaders poden tenir més funcions com per exemple llegir el que hi ha en el xip, és a dir extreure el programa del micro.

Hi ha molt diverses implementacions un bootloader sol residir a les primeres línies i al final de la memòria de programa. La memòria mostra un aspecte similar a la figura 3.3-1.

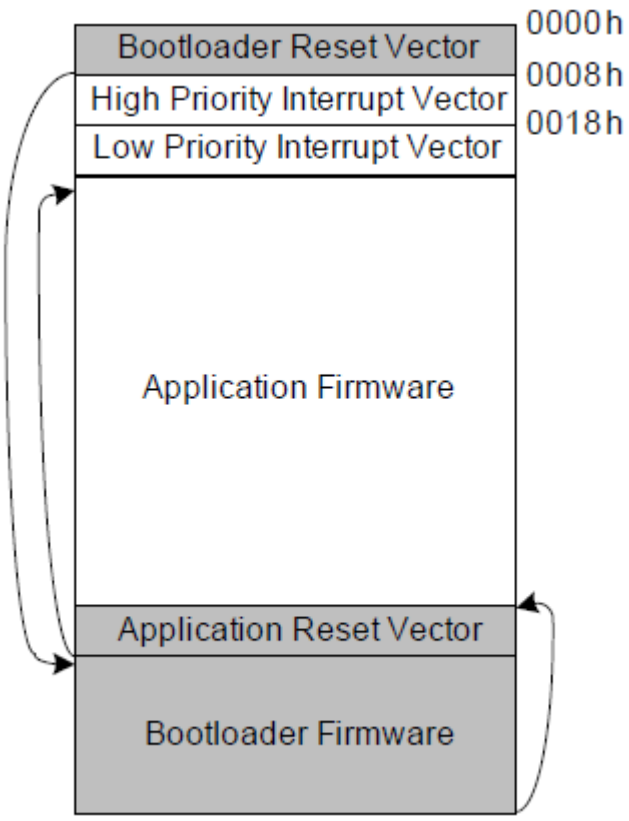


FIGURA 3.3-1

Veient la figura 3.3-1 s'entén que al engegar el microcontrolador les primeres instruccions (Bootloader Reset Vector) faran saltar l'execució fins on hi ha el codi real del bootloader (Bootloader Firmware) on es decidirà si cal rebre codi nou o saltar al llançador d'aplicacions (Application Reset Vector). Si cal rebre codi nou es gestiona la transferència amb l'ordinador s'escriu aquest nou codi al espai de programa i es salta a executar-lo. Si no hi ha codi nou es salta directament a executar el ja existent.

Podem veure els 2 vectors d'interrupcions (de fet el nostre microcontrolador fa servir un únic vector d'interrupcions) i per tant l'ús de bootloader no impedeix de cap manera la utilització del mecanisme d'interrupcions.

Com que sembla una bona eina per al desenvolupament es va decidir de fer servir un *bootloader* per al projecte. S'optà per l'AN1310 de la mateixa Microchip® ja que rep les comandes i programes per canal sèrie i s'anunciava com a compatible amb el PIC16F1827.

Es va intentar fer-lo funcionar *out-of-the-box*, no va funcionar. Microchip® ofereix el codi font de l'eina permetent que es modifiqui per a adaptar-lo a les necessitats del desenvolupador.

El codi ofert és en assemblador i per tant les adaptacions que es van anar fent-hi també es van fer amb assemblador. Les primeres modificacions van ser únicament per a adaptar els pins del software a els pins on hi ha realment els perifèrics en el PIC16F1827.

A la columna de l'esquerra es pot veure el codi tal com s'oferia i a la de la dreta les modificacions, hi ha el codi modificat íntegre a l'annex.

```
#define  PORTRX    PORTB          #define PORTRX  PORTB
#define RXPIN   2                 #define RXPIN   1

#define TRISTX  TRISB             #define TRISTX  TRISB
#define TXPIN   5                 #define TXPIN   2
```

Tot i que ara ja s'adreçava correctament a els pins on tenia connectat l'USART va caldre afegir algunes modificacions més. El codi original obviava la configuració del pins, possiblement perquè en models anteriors no eren multiplexables entre analògics i digitals i per tant no hi havia necessitat d'especificar el funcionament desitjada.

```
                          banksel ANSELB
                          bcf ANSELB, RXPIN
                          banksel TRISB
                          bsf     TRISB,     RXPIN
                          banksel PORTB
```

```
btfsc   PORTRX, RXPIN              btfsc   PORTRX, RXPIN
goto    AppVector                  goto    AppVector

btfss   PORTRX, RXPIN              btfss     PORTRX,  RXPIN
goto    $-1                        goto    $-1
```

També va ser necessari canviar la direcció d'alguns pins, de fet les instruccions ja hi eren a l'original però comentades:

```
;  bcf     TRISTX, TXPIN             bcf     TRISTX, TXPIN
   movlw   b'00100110'               movlw   b'00100110'
   movwf   TXSTA                     movwf   TXSTA
```

Com que amb totes les modificacions aplicades fins a aquest punt encara no s'havia aconseguit la connexió entre l'ordinador de sobretaula executant el client del bootloader i la MCU, vaig pensar que potser l'FTDI tenia algun problema que impedia que funcionés l'autonegociat del *baudrate.* Aquesta part del codi es va substituir completament per un codi que prefixava la configuració del port en sèrie a uns paràmetres coneguts, per tant la comparació costat per costat no tindria cap mena de sentit.

```
banksel SPBRGL
movlw .25
movwf SPBRGL
clrf SPBRGH
bcf BAUDCON, BRG16
bcf TXSTA, SYNC
bsf TXSTA, BRGH
```

Finalment amb totes aquestes modificacions el *bootloader* va connectar correctament amb l'ordinador de sobretaula, i permetia l'enviament de codis petits cap a la placa i també permetia llegir el contingut de la MCU per a desar-ho en un fitxer a l'ordinador.

Tot i que inicialment semblava funcionar perfectament a mesura que els codis s'anaven fent mes complexes les execucions començaven a comportar-se de forma estranya. En un primer moment vaig pensar que els comportaments estranys s'explicarien per una interferència del *bootloader* amb l'adreça destinades al vector d'interrupcions, però de seguida vaig veure que programes curts tant amb interrupcions com sense funcionaven i els programes més llargs no es comportaven correctament independentment de la presencia de codi per a tractar interrupcions.

Els comportaments indesitjables consistien en que l'execució es saltava erràticament les primeres instruccions del programa, no sempre un mateix nombre d'instruccions (per tant no es podia solucionar afegint un nombre definit de NOPs al inici del main). El comportament feia sospitar d'algun problema al saltar de bootloader cap a aplicació però ja m'havia excedit molt en el temps destinat a aquest component i es va decidir prescindir d'aquest component del programari en el projecte final.

3.3.2. PROGRAMARI DE CONTROL DESENVOLUPAT

En primer lloc és van escriure biblioteques per els diferents perifèrics que es farien servir en el projecte. A continuació es detallen aquestes biblioteques i la configuració que s'ha donat a cada un. El codi complet és accessible al annex.

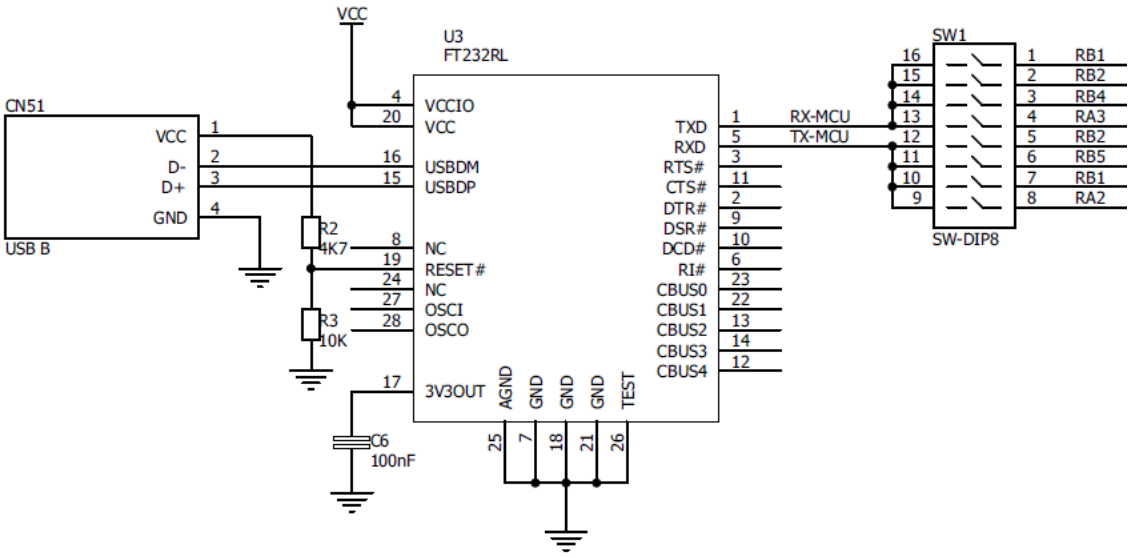
BIBLIOTEQUES DE PERIFÈRICS

Totes les biblioteques s'han intentar desenvolupar mantenint tant com ha estat possible la independència del maquinari, per tal que fossin aprofitables per a futurs desenvolupaments sense canvis o fent lleus adaptacions al codi existent.

USART

És el maquinari específic que permet la comunicació sèrie del PIC. Admet funcionar tant de forma síncrona com asíncrona i a diferents tasses de transmissió. Hem de tenir en compte que el nostre PIC no està

connectat directament a un port sèrie sino que passa a través de l'FTDI de la placa de desenvolupament. Aquest fet pot implicar algunes limitacions en l'ús del port.



ESQUEMA 3.3-1

Com podem veure a la part superior dreta del esquema 3.3-1 hi ha uns interruptors (SW-DIP8) que permeten decidir quins pins actuaran com a TX i RX, això dona flexibilitat al disseny i ens permet escollir pins que no estiguin en conflicte amb altres tasques que ha de dur a terme el microcontrolador.

També es pot veure que entre l'FTDI i el microcontrolador només es connecten les línies d'RX i TX per tant no es podrà fer servir el ***handshaking*** per maquinari. Tampoc hi ha senyal de rellotge, el que ens obliga a utilitzar el mode asíncron.

Per tant s'opta per fer servir els pins 1 i 2 del port B per a la transmissions per USART (RB1 és l'RX i RB2 el TX). La utilització d'aquests pins te una mínima interferència amb l'LCD ja que la línia RS de l'LCD està connectada al RB2 és un problema fàcilment evitable, simplement cal evitar fer transferències USART quan s'està transferint cap a l'LCD. A més a

més l'alternativa era utilitzar els pin RB5 per a TX, i aquest pin ja està connectat com a pin de dades de l'LCD.

Tenint en compte l'experiència prèvia amb el bootloader on l'autonegociacio de *baudrate* no va funcionar es va decidir fixar-lo a un valor acceptable. Microchip ofereix en els seus microcontroladors una unitat especifica per a l'USART, configurable a traves de registres especials. També ofereixen taules amb el valors necessaris per a obtenir diversos *baudrates* en funció de la freqüència de rellotge a la que s'està treballant. Es va optar per 19200, és estàndard i per tant tot el software d'ordinador l'oferiria com a possibilitat en la configuració. A més els valors que dona Microchip per a l'obtenció del *baudrate* no són totalment exactes, per qüestions de maquinari, hi ha un petit error. 19200 és una de les configuracions amb una tassa d'error més baixa (desviació de 0.16% respecte el valor teòric) de les que es poden obtenir amb un cristall de 8 Mhz.

CAPÇALERES

```
void initUART();
```

Inicialitza tots els registres necessaris per a l'enviament de dades a través de l'UART, cal executar-la sols una vegada i la configuració és persistent per a tots els enviaments.

```
void sendUART(char* data,int size);
```

Envia els tants bytes consecutius de dades com indica el paràmetre *size*, cal que les dades estiguin en memòria de dades. No és fa tractament d'errors, l'usuari és responsable de garantir que a partir de l'adreça indicada com a primer paràmetre hi ha *size* bytes consultables.

```
void printUART(const char* s);
```

Envia un string, és idònia per a passar missatges de text. Cal que tot el missatge es trobi en memòria de programa. No hi ha tractament d'errors, s'envien les dades fins que es troba un byte amb valor 0.

```
void printIntUART(int valor);
```

Converteix l'enter valor a una cadena ASCII i l'envia. Els valors s'interpreten com a enters no signats.

```
void printPromptUART(const char* prompt,int valor);
```

És una combinació de les dues funcions anteriors, permet enviar una cadena seguida d'un enter. Afegeix automàticament un salt de línia i retorn de carro després de l'enter. Cal que la cadena sigui constant i els valors s'interpreten com a enters no signats.

Tot i que l'USART és una interfície bidireccional no hi ha funcions per a la recepció de dades ja que no s'ha utilitzat en aquest projecte.

Per adaptar aquesta biblioteca a altres projectes cal modificar la funció d'inicialització per a que configuri el port com s'escau.

DISPOSITIUS ANALÒGICS

Són un conjunt de dispositius que permeten treballar amb tensions analògiques. Cal entendre que un microcontrolador és principalment digital i per tant aquestes unitats són necessàries si volem que pugui prendre mesures analògiques del món.

CONVERSOR ANALÒGIC DIGITAL

Si es connecta una tensió a un pin del microcontrolador i s'intenta llegir directament amb una instrucció com ara:

Int valor_llegit = RA0;

Obtindrem un valor digital, és a dir que no ens serà possible distingir si el valor entrant era 4.1 V o bé 5 V, simplement obtindré, un 1.

És molt important per a aquest projecte poder llegir tensions analògiques per a conèixer l'estat de la bateria i dels condensadors. Els microcontroladors acostumen a incorporar unitats de conversió (ADC) que ens faciliten aquesta tasca sense afegir components addicionals al circuit.

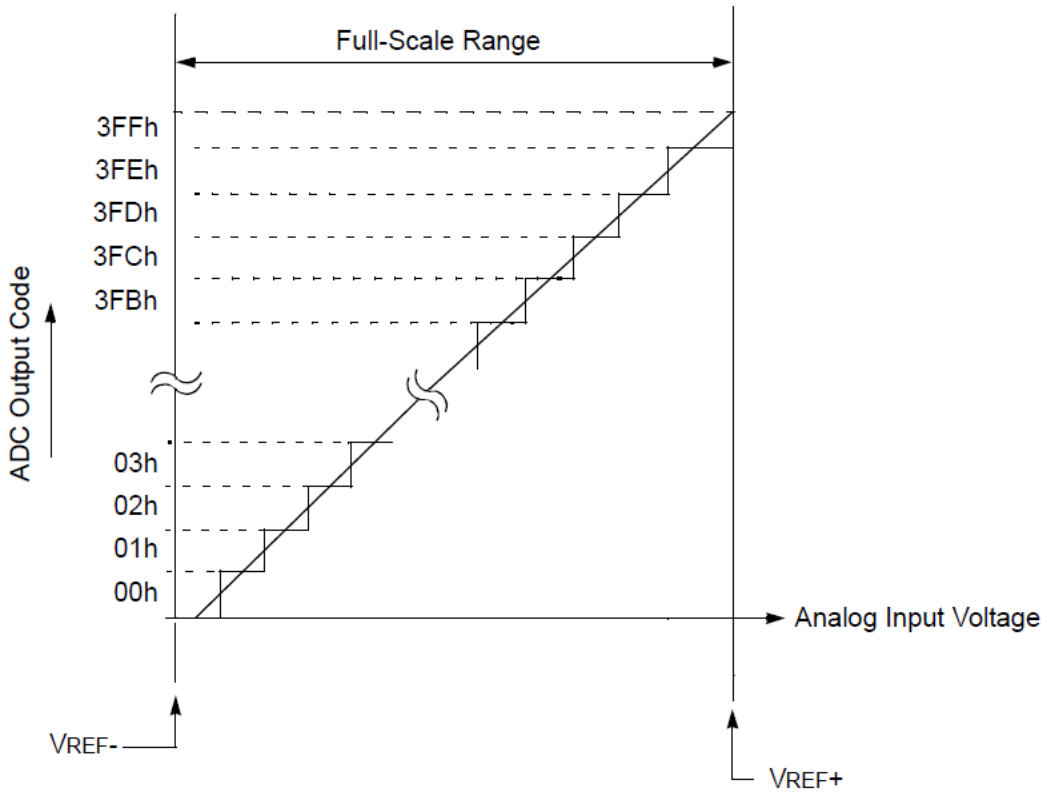


FIGURA 3.3-2

Hi ha diverses tecnologies per a aquesta operació, els conversos que Microchip ofereix en les seves MCUs és d'aproximacions successives. És a dir que va comparant el valor a mesurar amb d'altres i va ajustant la referencia respecte si és mes gran o més petit, cada pas d'aquesta aproximació li permet fixar un bit del valor resultant, l'ADC del PIC16F1827 és de 10 bits. És a dir que divideix el rang entre la referencia positiva i la negativa en 1023 parts i retorna en quina d'aquelles parts cau la tensió mesurada.

Cal entendre que com en tots els processos de digitalització de informació analògica hi ha una pèrdua de precisió. Naturalment si cal més precisió sempre es pot fer servir un mòdul de conversió de més bits i per tant capaç de fer distinció entre variacions mes petites en la tensió analògica.

Des d'un punt de vista pràctic el funcionament d'aquesta unitat és autònom, un cop inicialitzada i ha rebut l'ordre de conversió treballa

independentment del micro i pot alçar una interrupció per avisar que ja disposa del resultar. De fet amb una configuració especifica (fent servir com a rellotge un oscil·lador dedicat) és capaç de treballar fins i tot amb el microcontrolador adormit.

Aprofitant aquestes característiques per al projecte, sempre que s'emet una ordre de conversió al ADC s'adorm el micro. Tenint en compte que durant gran part del temps el microcontrolador es limita a esperar que es carreguin o descarreguin els condensadors aquest plantejament permet mantenir el micro adormit durant pràcticament tota l'estona amb l'estalvi d'energia que això comporta.

CONVERSOR ANALÒGIC DIGITAL

És el perifèric oposat a l'anterior, és capaç de generar una tensió analògica en funció d'un valor digital donat. També és força comú en els microcontroladors.

Tot i que no s'ha fet servir en el projecte final si que es va utilitzar durant l'etapa de desenvolupament per a generar tensions de prova per al ADC.

REFERÈNCIA DE VOLTATGES FIXADA

Hi ha un tercer perifèric analògic que permet generar voltatges de 1,024 V, 2,048 V i 4,096V com a referències tant per DAC com per a l'ADC. És molt útil ja que permet tenir referències mes baixes que la tensió d'alimentació sense haver d'afegir maquinari al disseny.

En el projecte es fa servir per a generar una referencia positiva per mesurar amb més precisió l'estat de la bateria.

CAPÇALERES

```
void readADC(char channel, char referencia_pos,
             char referencia_neg);
```

Aquesta funció inicialitza els registres de l'ADC i el configura amb els paràmetres donats a la crida. Dona l'ordre de conversió i adorm el microcontrolador.

```
void initDAC(char valor);
```

Configura el DAC per a que generi una tensió seguint l'equació:

$$Vout = \left(5 \cdot \frac{valor}{32}\right)$$

Cal que: $0 \leq valor \leq 32$

```
void initFVR(char valorDAC,char valorAD);
```

Inicialitza el mòdul d'FVR, les els paràmetres indiquen el valor a generar i cap a quin dispositiu dirigir-lo. En el projecte valorDAC sempre val 0, és a dir que el mòdul per al DAC està desactivat per a estalviar energia.

LCD

Com s'avança en el capítol de maquinari l'LCD és la principal via de comunicació amb l'usuari. La majoria de pantalles LCD no només tenen una interfície física comú sinó que al basar-se en el controlador HD44780 o compatibles també ofereixen una de lògica similar (poden haver-hi matisos en el procediment d'inicialització).

La biblioteca utilitza el mode de 4 bits (l'adaptador ens hi força) i assumeix que els bits de dades són els 4 bits alts del port B, tot i que amb petites modificacions es pot aprofitar per a altres projectes amb configuracions diferents.

L'habitual en els LCDs es fer servir un bus de dades de 8 bits, però per estalviar pins es poden fer les transferències en 2 blocs de 4. Aquesta dualitat de modes no afecta de cap manera el funcionament de l'LCD, independentment de com ens hi comuniquem es comporta de la mateixa manera i admet les mateixes comandes i dades, és transparent a l'usuari. S'ofereixen 2 cronogrames per a il·lustrar el funcionament dels 2 modes de transmissió.

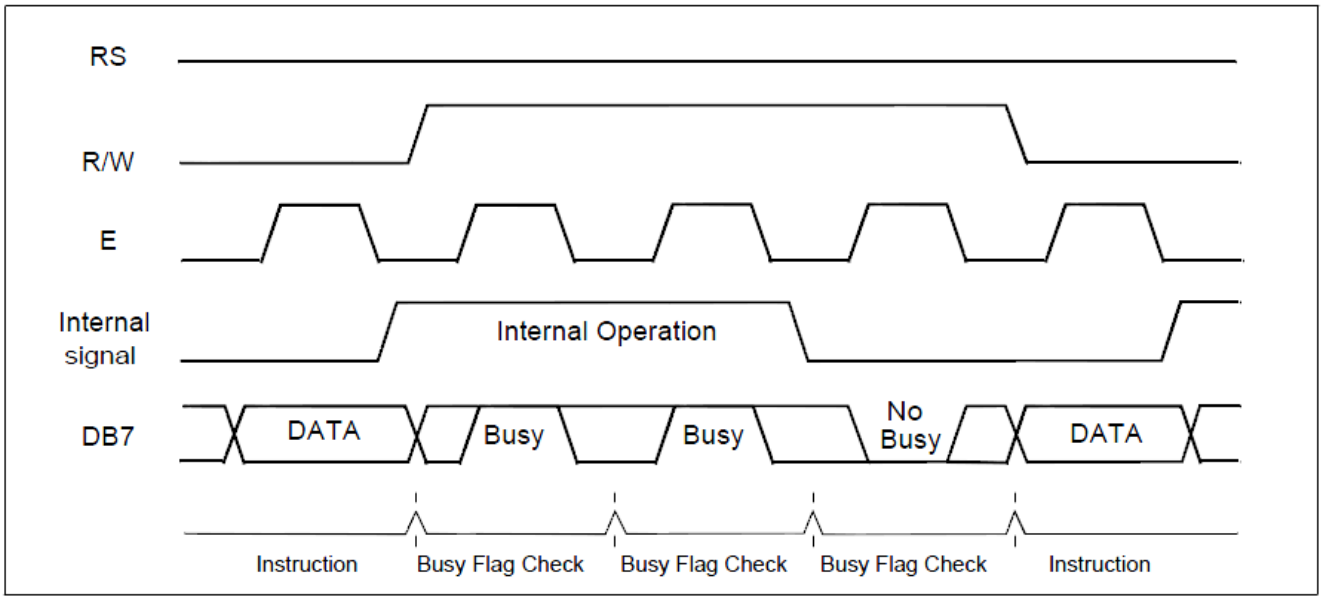


FIGURA 3.3-3 CRONOGRAMA MODE 8 BITS

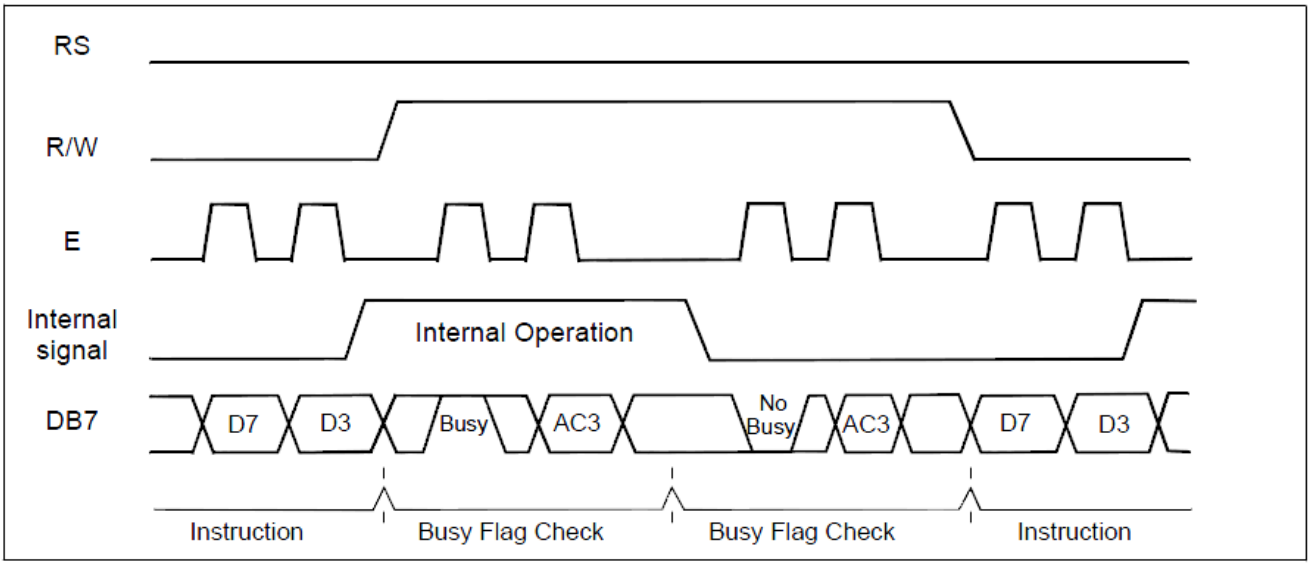


FIGURA 3.3-4 CRONOGRAMA MODE 4 BITS

Es pot veure en els cronogrames (3.3-3 i 3.3-4) que cal col·locar les dades o instrucció en el bus de dades i alçar el bit d'Enable. Si la transmissió és de 8 bits ja s'ha completat però si és de 4 cal carregar els 4 bits baixos al bus i tornar a alçar l'Enable per a indicar que cal llegir la segona part de la instrucció o dades.

Una altre de les limitacions que imposa el cablejat de l'LCD amb el PIC (detallat en el capítol de maquinari) és que la línia d'RW està fixada a 0 (GND) per tant sempre està en mode lectura, per aquest motius no s'han implementat funcions d'escriptura de la pantalla i s'han substituït les comprovacions de l'indicador d'ocupació (que requereixen del mode de lectura) per retards que garanteixen que l'LCD te temps a acabar l'operació.

Tot i que es va afegir maquinari per a poder desconnectar l'LCD quan fos necessària per estalviar energia d'aquesta funció no s'en encarrega aquesta biblioteca ja que considero que és una peculiaritat del projecte i que, per tant, no és exportable a altres solucions.

CAPÇALERES

```
void initLCD();
```

És la funció d'inicialització de l'LCD, segueix estrictament al descripció del fabricant per al procés. S'utilitza la transmissió de 4 bits, autoincrement del cursor i mode de 2 linies de pantalla. Té l'efecte col·lateral de netejar la pantalla i tornar el cursor a la posició 0,0.

```
void lcdPutc(char c);
```

Envia el caràcter c, com a dada, a l'LCD. Si és representable la pantalla el mostrarà a la posició on hi ha el cursor en aquell moment.

```
void lcdCommand(char c);
```

Envia el byte c, com a comanda, a l'LCD. A la capçalera s'han definit constants per les comandes de tornar a la posició 0,0 i per a la neteja de la pantalla.

```
void lcdSetDDRAM(char c);
```

Fixa l'adreça de memòria on s'estan enviant dades a la LCD, permet fixar la posició del cursor (que al cap i a la fi consisteix en especificar una adreça).

```
void lcdGotoXY(char x, char y);
```

Permet especificar a quina posició es vol el cursor, de forma cartesiana, tot i que tècnicament fa el mateix que l'anterior aquesta és molt mes amigable a l'usuari.

```
void lcdPrint(char* s);
```

Imprimeix una cadena de text, acabada en un byte a 0. Cal que la cadena sigui en memòria de dades. No fa comprovació d'errors, és responsabilitat del usuari garantir que s apunta a una posició amb les condicions necessàries.

```
void lcdPrintConst(const char* s);
```

Ídem a l'anterior però per a cadenes en espai de programa.

```
void lcdPrintInt(int valor,int pos_x,int pos_y);
```

Converteix a ASCII el valor i l'imprimeix per la pantalla. La posició és on quedarà la xifra de menys pes, i s'escriu de dreta a esquerra. Els nombres s'interpreten com enters sense signe.

```
void lcdPrintLong(unsigned long valor,int pos_x,int pos_y);
```

Té el mateix funcionament que l'anterior però admet longs com a paràmetre a imprimir.

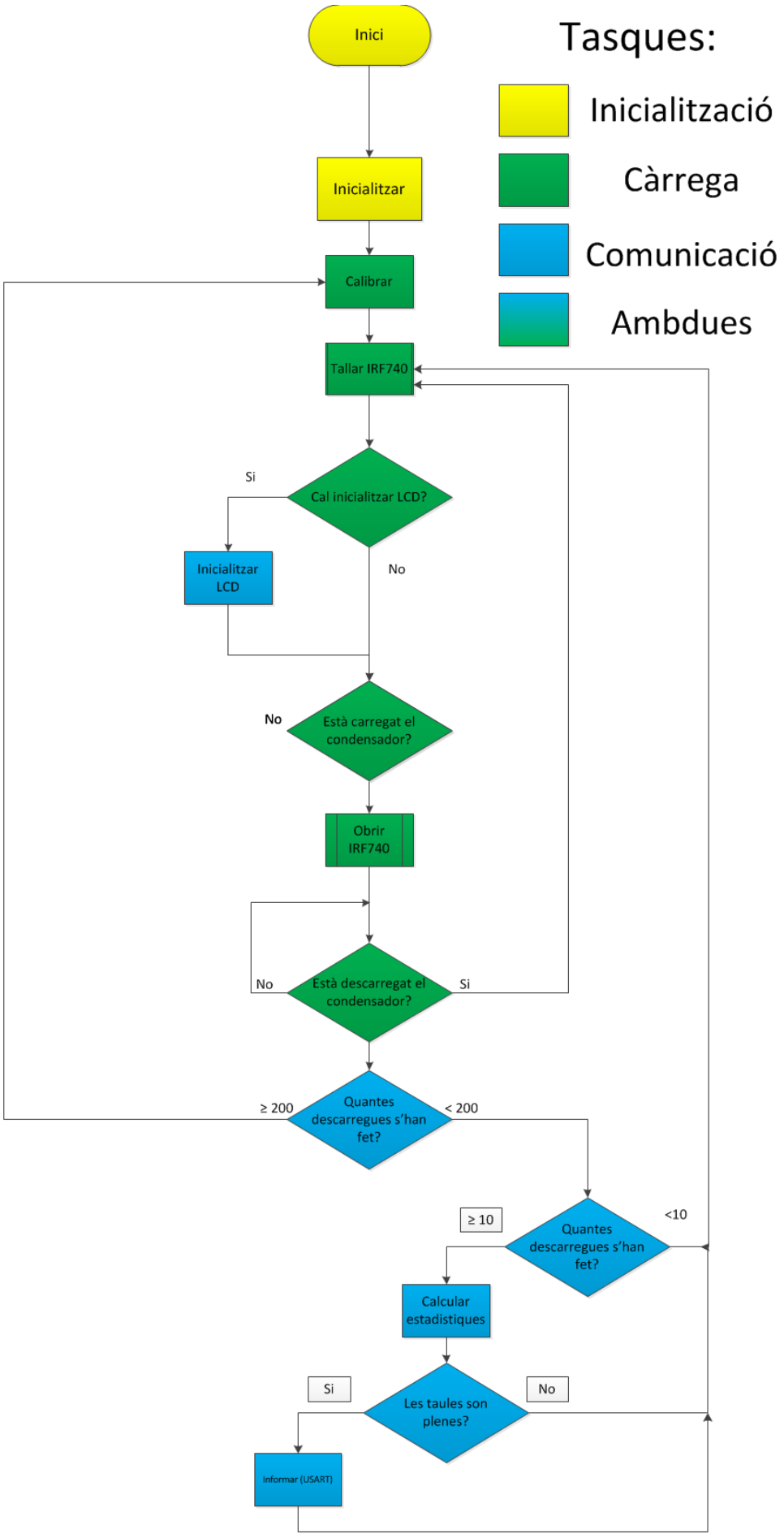
PROGRAMA DE CONTROL

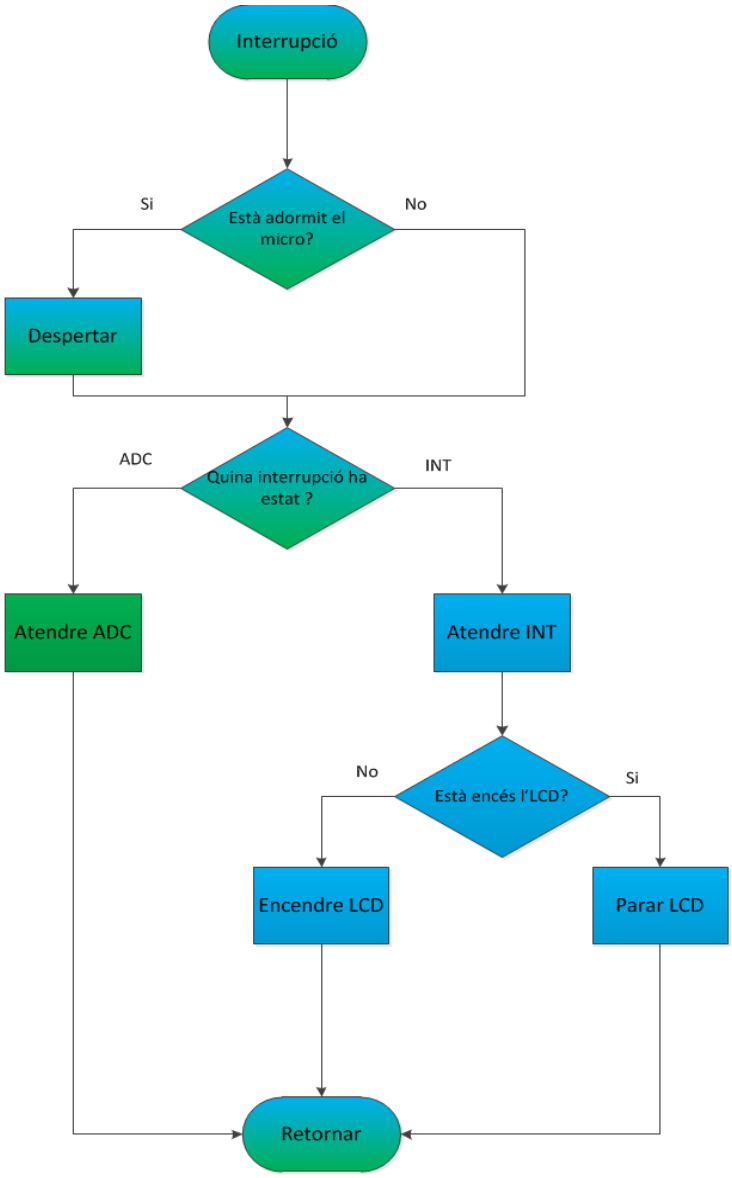
El programa de control o principal s'ha concebut com una màquina d'estats finita, de manera que no es passa al següent estat fins que no s'ha completat l'actual. Podem veure aquest plantejament en el *main* del programa.

```
void main(){
    init();
    while(1){
        lcd_usage();
        if (estat == 0) calibrar();
        if (estat == 1) esperarCarrega();
        if (estat == 2) descarrega();
        if (estat == 3) estadistiques();
        if (estat == 4) informar();

        if (descarregues >= 200 && estat==1){
            descarregues = 0;
            estat=0;
        }
    }
}
```

El codi complet del programa es troba als annexos. A continuació s'ofereix el diagrama de flux de l'aplicació.


Inici
Inicialitzar
Calibrar
Tallar IRF740
Cal inicialitzar LCD?
Si
No
Inicialitzar LCD
Està carregat el condensador?
No
Obrir IRF740
Està descarregat el condensador?
No
Si
Quantes descarregues s'han fet?
≥ 200
< 200
Quantes descarregues s'han fet?
≥ 10
<10
Calcular estadistiques
Les taules son plenes?
Si
No
Informar (USART)
Tasques:
Inicialització
Càrrega
Comunicació
Ambdues



Tasques:

Inicialització

Càrrega

Comunicació

Ambdues

En els diagrames podem veure que just a l'inici de l'execució s'inicialitza el maquinari. Aquest procés engloba:

- La configuració dels ports, és a dir especificar quins pins són entrades i quins són sortides. I fixar quins són analògics i quins digitals.
- Netejar les taules d'estadístiques.
- Donar valor a els pins de control dels 2 transistors (principal i control de 'LCD).
- Inicialització de l'USART
- Inicialització del FVR.

És un estat especial , ja que és l'únic estat a el que no es torna sota cap circumstancia en una execució.

Cal saber fins a quin punt s'han de carregar els condensadors o fins a quin punt es descarreguen (ja que naturalment només baixen fins al nivell de la bateria). Si fixéssim en el codi aquests valors res funcionaria adequadament. Podríem estar esperant a que el condensador es carregues fins un valor impossible en les condicions del moment o bé podríem estar tallant la carrega abans d'hora a cada cicle. És necessari un procés de calibració que pugui fixar aquests valors a cada execució.

Aquest procés consisteix en deixar carregar els condensadors durant 4 segons (el micro es manté adormit durant aquest temps) i mesurar el valor que han assolit. S'entén que 4 segons per a les capacitats amb que treballem és un temps suficient i que s'haurà assolit el màxim. A aquest valor màxim es redueix en 5, ja que els condensadors es carreguen de forma asimptòtica i carregar aquests últims milivolts a cada iteració és molt lent, i s'emmagatzema el valor com a llindar superior. Llavors es tanca el transistor principal durant 200ms i es fa un procediment simètric per obtenir el llindar inferior. A partir d'aquest punt ja entrem en l'operació habitual del disseny on es van iterant obertures i tancaments controlats del transistor per a carregar la bateria.

Les condicions són canviants, la bateria va carregant-se i per tant el llindar inferior no és constant i la radiació solar pot augmentar o reduir-se (per exemple passa un núvol) modificant el llindar superior. Per aquest motiu cada 200 cicles de càrrega-descàrrega es torna a llançar el procediment de calibració.

Considero que la resta de estats són prou autodescriptius. S'han omès alguns detalls per a facilitar la comprensió dels diagrames.

Detalls d'implementació

Tot i que en l'apartat anterior s'ha detallat el funcionament general de l'aplicació m'agradaria detallar-ne alguns punts.

Watchdog

El *Watchdog Timer* és un temporitzador especial pensat per a controlar el funcionament correcte d'una aplicació. El seu ús habitual consisteix en prefixar-li un interval de temps i deixar-lo córrer, quan arriba a l'overflow reinicia l'execució. Aquest comportament tot i que una mica estrany pot resultar molt útil, un programa en el seu transcurs normal ha d'anar tornant a 0 el WDT (amb una instrucció assambler especifica) abans que aquest produeixi un reset del sistema. D'aquesta manera sabem que mentre el programa esta en la zona habitual , és a dir correcte, d'execució s'anirà posant a 0 periòdicament i no es produirà el reset. Si algun esdeveniment inesperat descontroles l'execució o be quedes atrapada en un bucle infinit es deixaria de posar a 0 i per tant al cap d'un *overflow* es reiniciaria el sistema tot retornant-lo a un estat controlat.

Aquest ús habitual del WDT no s'aplica en el projecte, ja que inclou esperes llargues (carrega i descarrega del condensador). S'hi es poses la instrucció de neteja de WDT dins d'aquests estats es perd la protecció contra esperes infinites i si no es poses en aquests estats es corre el risc de reiniciar l'aplicació quan realment esta funcionant amb normalitat. El PIC no coneix la capacitat dels condensadors i per tant no pot saber si estan

trigant massa a omplir-se i fixar un valor d'espera limitaria la flexibilitat del sistema.

El WDT té una altre peculiaritat que resulta útil per al projecte. És capaç de funcionar mentre el processador dorm i a més a més si el *timeout* es produeix mentre està dormint el processador no es reinicia la execució sinó que simplement es desperta el micro. Aquest funcionament s'aprofita per a temporitzar parts de l'execució, concretament els 4 segons (fent ús del WDT estès propi d'un PIC16F18xx) d'espera de la calibració es controlen amb el WDT. En aquest projecte mentre el sistema està despert el WDT no està en funcionament.

TIMEOUT D'ESTAT

Tenint en compte que no es podia fer servir el WDT per evitar esperes infinites per els motius exposats en l'apartat anterior s'ha simulat la seva funció amb codi. S'ha afegit un comptador de cicles tant a l'espera de càrrega com a la descàrrega i s'ha ajustat experimentalment per a les capacitats mes altes. Quan s'excedeix aquest timeout es torna a calibrar el sistema ja que s'entén que hi ha algun problema amb els llindars. Aquesta recalibració no es produiria mai per la via habitual de les 200 descarregues perquè si hi ha un problema no es produeixen descàrregues.

RECOLLIDA DE DADES

Cada 10 cicles es passa per un estat més que s'encarrega d'omplir una taula de valors per a poder calcular la mediana de l'estat del condensador i la bateria. Es va descartar la mitjana ja que es un indicador estadístic feble (afectat per *outliers)* i al inici de les execucions donava valors falsos.

TRACTAMENT DE L'LCD

Quan l'usuari prem el polsador es genera una interrupció externa (INT) que està habilitada (excepte durant l'estat de calibració per evitar interferir en el còmput de 4 segons). Aquesta interrupció redirigeix la execució cap a la seva rutina. És una rutina molt senzilla, si l'LCD està encès

el para, si està parat l'encén i alça un bit. La rutina d'interrupció només s'encarrega de commutar el transistor del LCD perquè l'usuari vegi immediatament que s'ha produït el canvi que ell demanava. La inicialització i enviament de dades cap a la pantalla ja es faran a la propera iteració del main, cal tenir en compte que tal com està ideat el programa mai es triga més d'uns milisegons a iterar altre cop a el main. Amb això aconseguim donar sensació d'immediatesa a l'usuari i evitem executar funcions lentes (inicialització de la pantalla) dins la rutina d'interrupcions.

4. RESULTATS

Ara ja tenim tant el maquinari com el programari necessaris per a fer funcionar la solució. Cal recollir dades per comprovar que realment es compleixen els objectius del projecte.

4.1. ENTORN DE PROVES

És van carregar la bateria durant 4 hores sota llum solar d'intensitat silimar (no es pot garantir que dos dies siguin idèntics) i amb condensadors C1 de diferents capacitats. La càrrega s'iniciava sempre amb la bateria a 9 V ja que no és lineal i calia garantir que els resultats fossin comparables entre si. Les mesures es van prendre tant amb un voltímetre com el dispositiu (recordem que el polsador fa aparèixer per pantalla l'estat de la bateria). Es prenien mostres cada 10 minuts, la pantalla es deixava aturar per *timeout* de manera que en totes les proves va estar operativa una mateixa estona.

Es presenten dues gràfiques per a cada condensador, corresponent a la mesura presa amb voltímetre i la mesura presa per el PIC. Hi ha un petit *offset* entre les dades recollides per el PIC i les del voltímetre que es pot explicar per els díodes que hi ha al regulador d'alimentació de la placa de desenvolupament i que creen un desfàs entre el GND de la bateria i el del PIC. L'eix horitzontal representa intervals de 10 minuts

4.1.1. CONDENSADOR PETIT

Instant	mV	V
1	8863	9
2	9229	9,22
3	9519	9,51
4	9732	9,72
5	9901	9,9
6	10038	10,04
7	10160	10,19
8	10251	10,25
9	10361	10,32
10	10420	10,4
11	10480	10,48
12	10542	10,52
13	10573	10,57
14	10620	10,61
15	10648	10,66
16	10679	10,68
17	10711	10,71
18	10726	10,73
19	10742	10,74
20	10773	10,77
21	10786	10,79
22	10786	10,81
23	10801	10,82
24	10817	10,83
25	10833	10,85

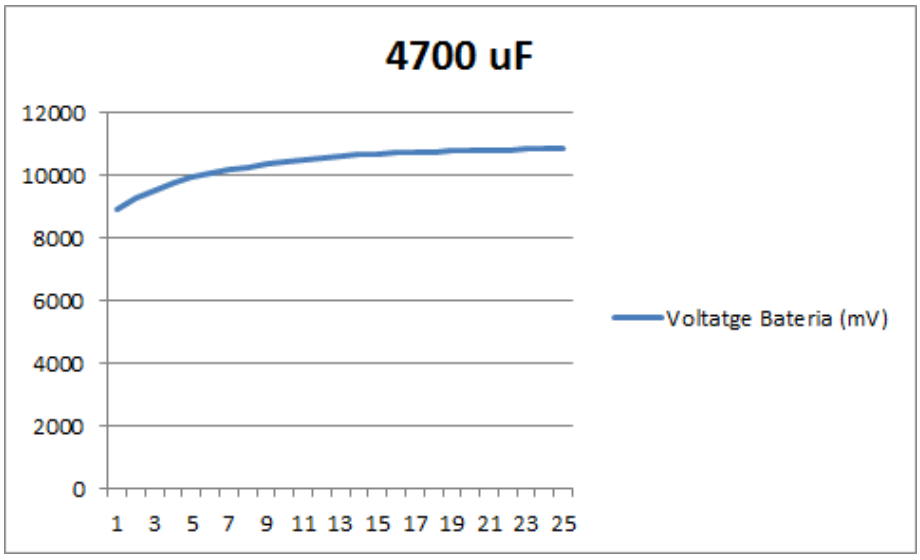


FIGURA 4.1-1

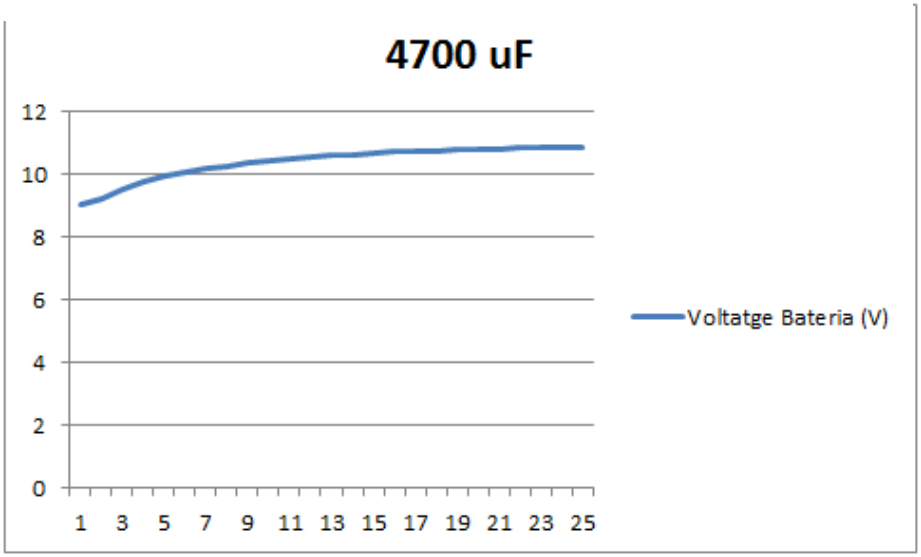


FIGURA 4.1-2

La càrrega és ràpida al principi del procés tot i que va alentint-se a mesura que ens acostem al valor màxim de la bateria.

4.1.2. CONDENSADOR MITJÀ

Instant	Placa (mV)	Tester (V)
1	9028	9
2	9382	9,57
3	9838	9,82
4	10054	10,02
5	10207	10,16
6	10314	10,29
7	10389	10,39
8	10498	10,47
9	10542	10,55
10	10630	10,6
11	10648	10,64
12	10695	10,68
13	10726	10,72
14	10742	10,74
15	10773	10,77
16	10786	10,79
17	10801	10,81
18	10817	10,83
19	10833	10,85
20	10848	10,86
21	10848	10,88
22	10864	10,89
23	10880	10,9
24	10895	10,91
25	10896	10,91

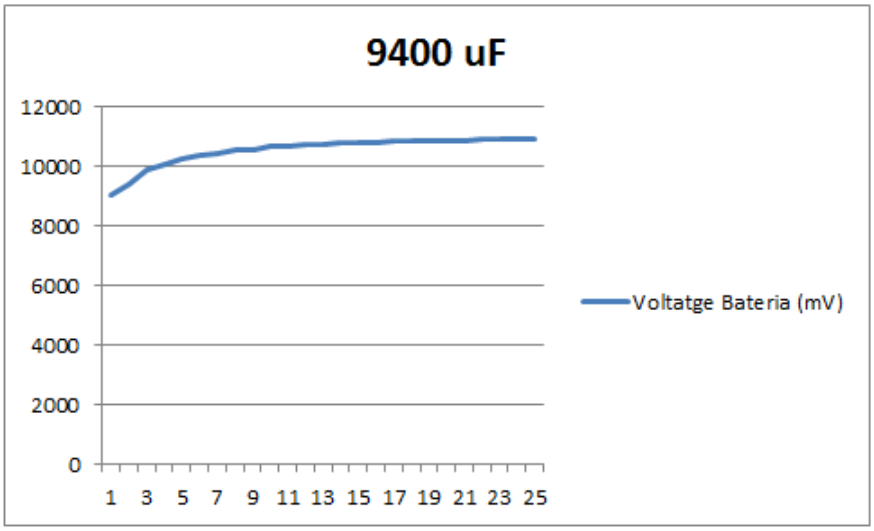


FIGURA 4.1-3

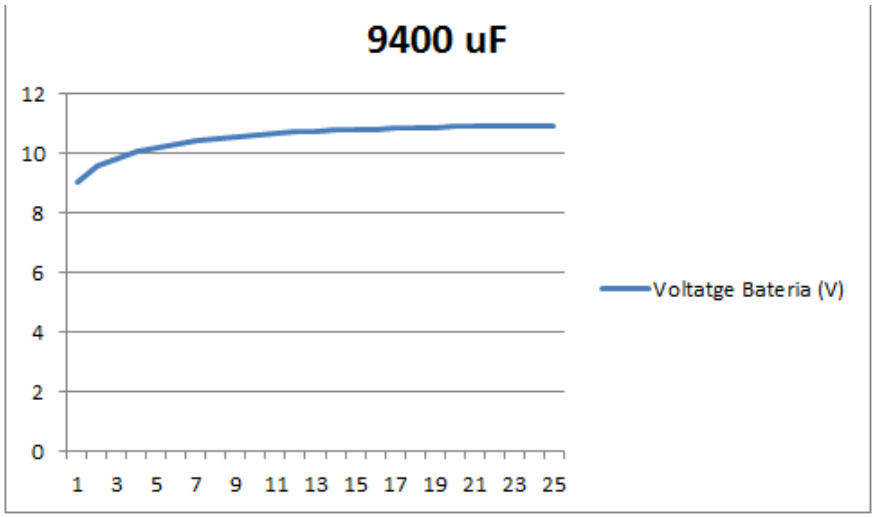


FIGURA 4.1-4

Tot i que el pendent de la fase inicial és més pronunciat que l'anterior es pot veure que també es va alentint el procés de càrrega a mesura que avança.

4.1.3. CONDENSADOR GRAN

Instant	Placa (mV)	Tester (V)
1	8969	9
2	9122	9,05
3	9275	9,29
4	9515	9,5
5	9745	9,69
6	9941	9,89
7	10082	9,98
8	10192	10,09
9	10251	10,16
10	10314	10,24
11	10420	10,35
12	10467	10,42
13	10526	10,48
14	10620	10,56
15	10648	10,59
16	10711	10,65
17	10742	10,71
18	10755	10,73
19	10770	10,74
20	10786	10,75
21	10801	10,78
22	10817	10,81
23	10833	10,82
24	10848	10,86
25	10864	10,86

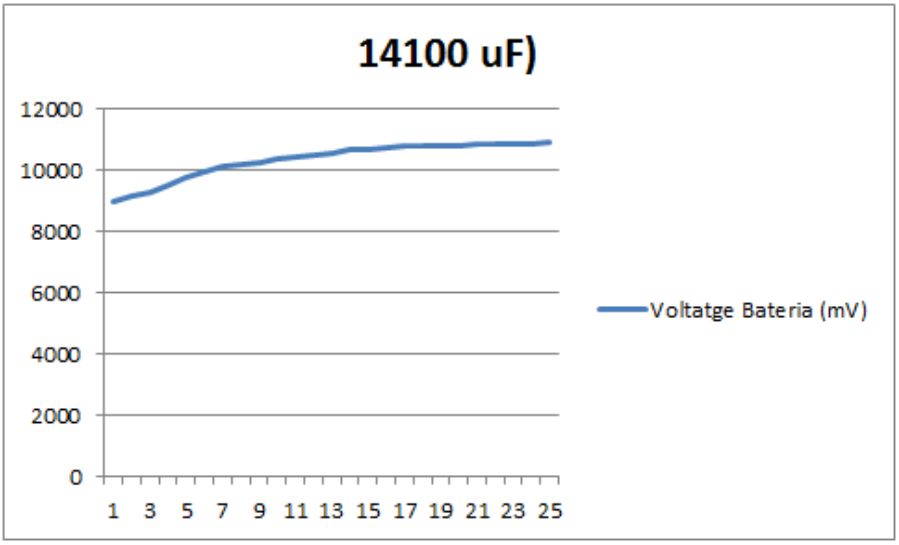


FIGURA 4.1-5

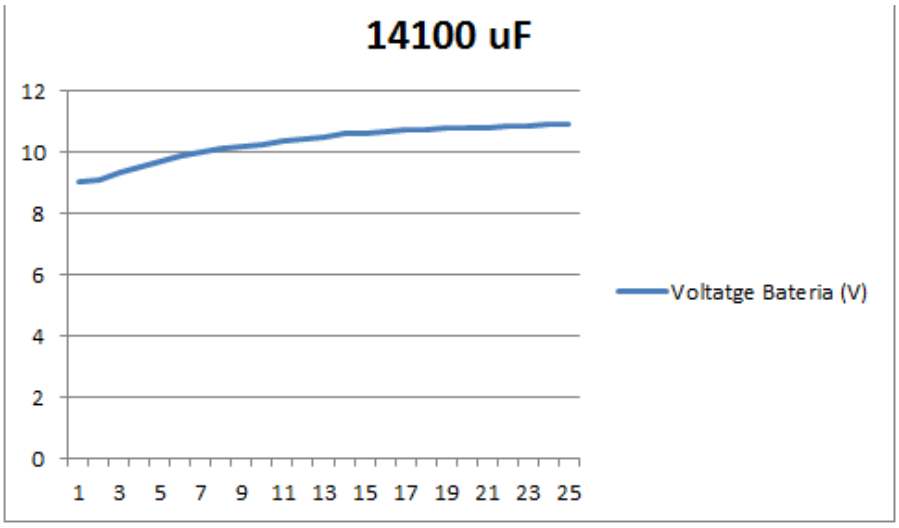


FIGURA 4.1-6

Tot i que pot semblar poc intuïtiu la pendent inicial de càrrega és menys pronunciada que en les mesures anteriors. Treballar amb grans capacitats implica un període de càrrega més elevat i per tant menys descàrregues per unitat de temps. Aquest matís podria explicar aquesta diferencia en el pendent.

4.1.4. CONTRAST DE RESULTATS

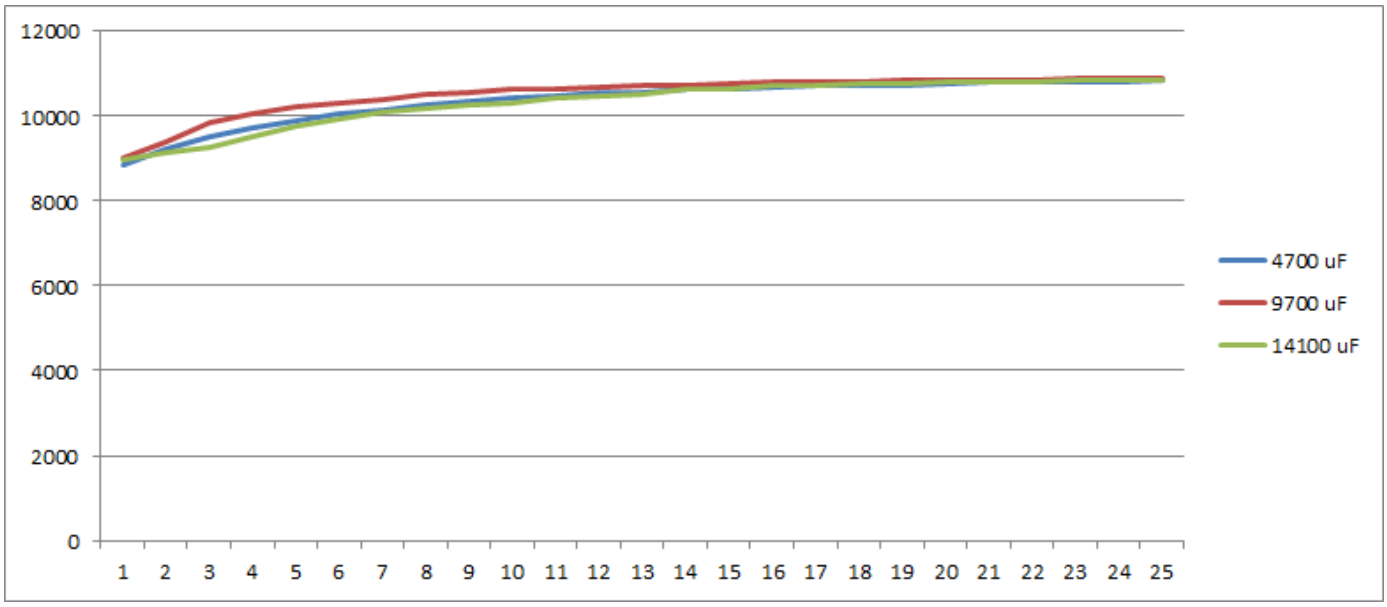


FIGURA 4.1-7

Cal dir que disposem de poques dades per a poder considerar-les estadísticament vàlides. Aquesta interpretació de les dades és, per tant, preliminar i caldria corroborar-la o desmentir-la amb més observacions.

Es veu clarament que l'inici de la càrrega és molt més ràpid que la segona meitat del procés. Aquest fenomen no és exclusiu del mètode de càrrega d'aquest projecte de fet si es carreguen les bateries amb un carregador exprés que s'alimenti de la presa de corrent alterna domèstica també es pot observar aquest alentiment de la càrrega.

No és clar si realment la capacitat del condensador afecta al procés, caldria fer moltes més proves i possiblement un estudi ANOVA per determinar si la diferència entre condensadors explica una diferència entre els pendents de càrrega.

5. CONCLUSIONS

En aquest capítol s'han analitzat els diversos aspectes amb m'he anat trobant durant l'elaboració d'aquest projecte.

5.1. ANÀLISI ECONÒMIC

Cal tenir en compte que hi ha dos conceptes molt diferents a valorar en aquest apartat: l'un és el cost del desenvolupament del projecte i l'altre el cost unitari de cada producte (cada regulador).

5.1.1. DESENVOLUPAMENT DEL PROJECTE

Programari:

MPLAB 8.60	Gratuït
HI-TECH C Compiler 9.50 Lite	Gratuït
Serial Input Output Monitor	Gratuït
Labcenter Electronics Proteus	200 €

Components:

PIC16F1827-I/P	1.30 €
PIC-Ready2	21,86
Adaptador LCD	7,55 €
Pantalla LCD	7,39 €
IRF740	0,80 €
BD440	0,70 €
Condensador 4700 uF	1,10 €
Altres Components	2,50 €

Desenvolupament:

Analisi i disseny	60 h	50 €/h	3000 €
Desenvolupament	320 h	45 €/h	14400 €
Prototipatge	30 h	30 €/h	900 €
Documentació	50 h	35 €/h	1750 €

Cost total del desenvolupament del projecte: **7333.2 €**

5.1.2. COST UNITARI

Assumint que es cedeix gratuïtament el coneixement tècnic del projecte. També considerem que les versions a desplegar no es basaran en una placa de desenvolupament sinó que el PIC anirà muntat directe sobre placa.

PIC16F1827-I/P	1.30 €
Pantalla LCD	7,39 €
IRF740	0,80 €
BD440	0,70 €
Altres Components	5,00 €
Muntatge	20 €

S'ha omès el cost del panell solar i la bateria ja que s'entén que són costos variables en funció dels models.

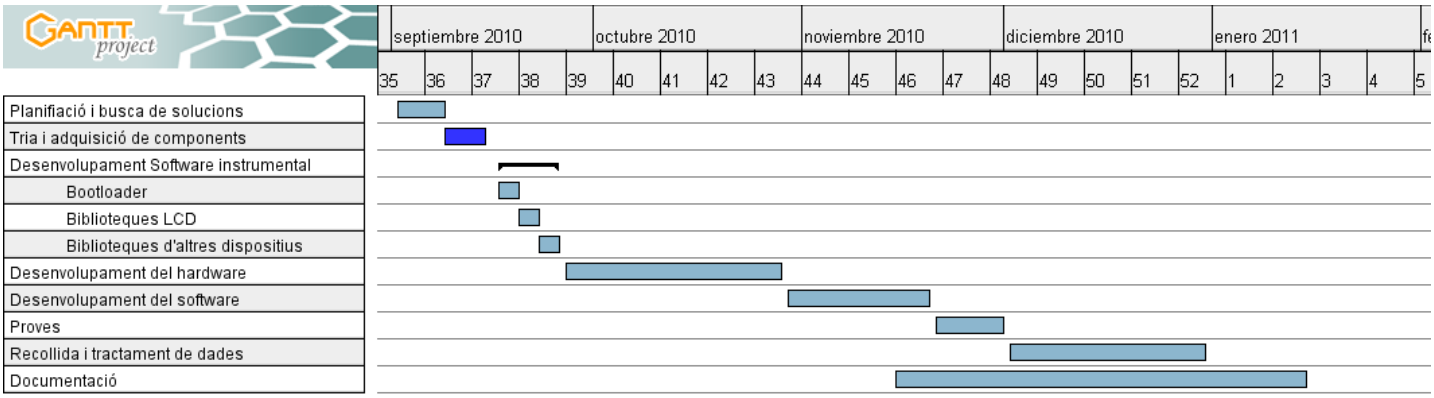
Cost d'unitat desplegada: **33.89 €**

És un preu assumible, i que molt possiblement es pugui ajustar molt si es produïssin en massa els reguladors. Amb aquest preu donem per

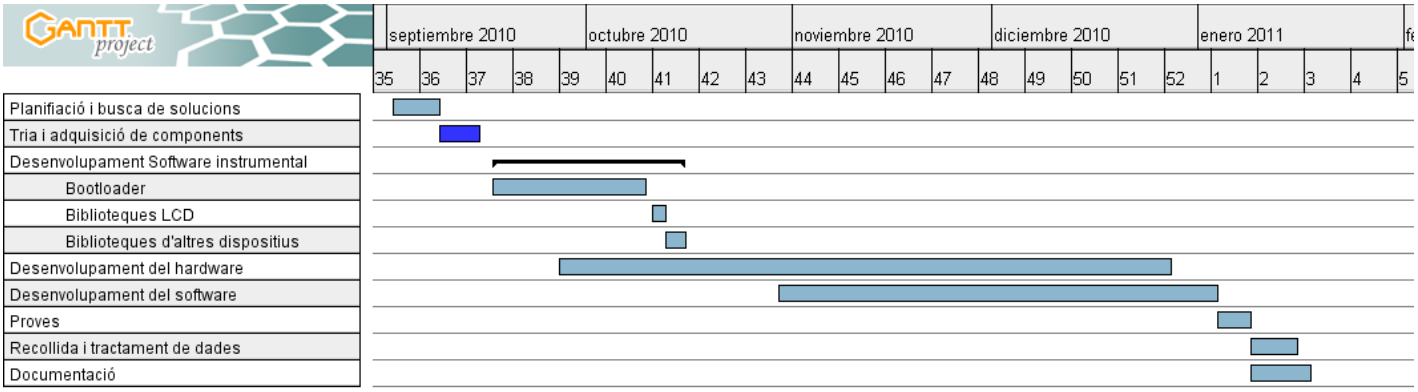
acomplert l'objectiu de desenvolupar una solució de baix cost. A més a més els components utilitzats són força habituals i és troben amb facilitat.

5.2. PLANIFICACIÓ

La planificació inicial del projecte és la mostrada en el següent Gantt:



Tot i que el projecte ha seguit el següent diagrama de Gantt:



Es pot veure que el desenvolupament del bootloader (detallat en el capítol de programari) va ser molt més llarg de el que s'esperava inicialment, i va endarrerir el temps de desenvolupament de la resta de projecte.

El desenvolupament del maquinari també va ser molt més llarg de el que s'esperava inicialment. Hi va haver molts imprevistos durant el procés i per això el temps que es va preveure per a aquesta tasca dista tant de la realitat. Aquests imprevistos en el maquinari van fer endarrerir tot el

projecte. En especial se'n han ressentit les fases de proves i recollida de dades.

5.3. ANÀLISI D'OBJECTIUS

Es pot dir que s'ha assolit la majoria dels objectius del projecte. I per tant m'atreviria a dir que ha estat un èxit.

5.3.1. GENERALS

S'han complert els objectius generals dels projecte ja que s'ha obtingut un maquinari autònom funcional i de baix cost. Considero important mencionar els bons resultats de càrrega en el temps (1.5V en 4 hores) i també és interesant l'aparent manca de relació entre la càrrega a mitg plaç i la capacitat de l'acumulador.

5.3.2. ESPECÍFICS

Tot i que el resultat final del projecte ha estat positiu, per qüestions de temps no s'han pogut recollir totes les dades que m'hauria agradat i per tant no és possible fer un estudi detallat sobre el comportament de tecnologia.

5.3.3. ACADÈMICS

Els objectius acadèmics del projecte s'han assolit. Durant el projecte he aprés a soldar i nous conceptes d'electronica. A més he pogut experimentar amb els diferents perifèrics d'un PIC i amb el seu mode de baix consum (dormir). Tot i que el desenvolupament del bootloader no ha estat exitós m'ha servit per a millorar els meus coneixements en assambler de PIC.

5.4. CONCLUSIÓ PERSONAL

Primerament vull admetre que quan se'm va proposar aquest projecte era força reticent a acceptar-lo perquè el fort component de maquinari em tirava enrere. Tot i això, lògicament, vaig decidir acceptar-lo i no me'n penedeixo perquè he aprés molt durant aquest projecte.

L'he trobat interessant des de diferents vessants. Per una banda és el primer ús real que faig de sistemes encastats, si bé no és la primera vegada que hi treballo fins ara sempre havien estat practiques preparades i sobre una placa d'entrenament. La vessant més electrònica del projecte m'ha permès acostar-me a un camp que d'altre manera molt difícilment hauria explorat.

Tot i que no s'han aconseguit tantes dades com m'hauria agradat la meva valoració del projecte és positiva.

He aprés la importància d'una bona planificació, realista amb les tasques a realitzar. La planificació original era molt optimista i a més no es va seguir massa rigorosament, de manera que el bootloader i els problemes de maquinari van acabar imposant-se sobre la resta del projecte.

5.5. LÍNIES DE FUTUR

Aquí es recullen algunes idees que podrien ser interessants per a desenvolupar en un futur.

- Recollir més dades i fer un estudi en profunditat dels resultats. Tot i que aquest projecte ha demostrat que la idea és viable i funcional seria interessant disposar de més dades per a poder modelitzar i molt possiblement millorar el funcionament d'aquest disseny.
- Adaptar el maquinari i desenvolupar el programari sobre una plataforma oberta (per exemple Arduino), això garanteix fer-ho accessible a molta mes gent.
- Fer les adaptacions necessàries per a poder aprofitar altres fons d'energia renovables per a carregar bateries.
- Desenvolupar una aplicació per a ordinador que reculli les dades del microcontrolador i permeti tractar-les, potser fins i tot fer que el sistema sigui governable remotament des de internet.

6. GLOSSARI

En aquest capítol s'han recollit alguns termes que apareixen en aquest document.

ADC: *Analog to Digital Converter* (Conversor d'analògic a digital). És un dispositiu capaç de mostrejar tensions i convertir-les a un valor digital per a poder-hi treballar. N'hi ha de autònoms tot i que és molt freqüent que un microcontrolador n'incorpori.

DAC: *Digital to Analog Converter* (Conversor de digital a Analògic). És un dispositiu capaç de generar una tensió proporcional a un valor digital que se li dona com a entrada. Molt microcontroladors n'incorporen.

FTDI: *Future Technology Devices International.* És el nom d'una companyia especialitzada en xips conversos entre RS232 (port sèrie) i USB. Aquestes sigles també designen la majoria dels seus productes. Per a totes les aparicions del terme en el text es fa referencia al producte i no a la companyia.

FVR: *Fixed Voltage Reference* (Referència de voltatge fixada). És un component que incorporen alguns microcontroladors capaç de generar una tensió constant independentment de l'alimentació que se li subministri (lògicament cal mantenir uns paràmetre d'alimentació compatibles amb el dispositiu).

IDC10: Nom comercial amb el que Mikroelektronica referencia els connectors 5+5.

LCD: *Liquid Crystal Display* (pantalla de cristall líquid).

LED: *Light-emitting diode* (díode emissor de llum).

MCU: *Microcontroller Unit* (Unitat microcontroladora). Vegis microcontrolador.

Microcontrolador: És un petit ordinador, muntat sobre un únic circuit imprès. N'hi ha molts fabricants i models de manera que hi ha un ampli ventall de dispositius disponibles. Són totalment autònoms , un cop programats no necessiten cap sistema informàtic i si reben l'energia elèctrica necessària funcionaran d'acord amb el seu programa intern.

PIC: Actualment Microchip ja no utilitza aquest nom com a acrònim. És un dels microcontroladors fabricats per Microchip. Vegis microcontrolador.

ROM: *Read only memory* (memòria de només lectura).

***Trainer*:** Placa d'entrenament. És una placa de desenvolupament per a microcontroladors que ja inclou perifèrics per a simular diverses entrades i sortides. Estan pensades per a la formació.

USART: Universal syncronous/asynchronous receiver/transmitter (receptor/transmissor universal asíncron) és el mòdul encarregat de proporcionar connectivitat sèrie a l'MCU. És freqüent referir-si com a UART en casos on s'utilitza únicament el mode asíncron.

WDT: *Watchdog Timer* (Temporitzador de gos guaita). És un dispositiu, normalment integrat en un microcontrolador, capaç de ordenar un reinici d'un altre equip si no se li envien senyals de control regularment. El seu ús més habitual és garantir el correcte funcionament d'un microcontrolador.

XLP: *eXtreme Low Power . Nom comercial de la tecnologia de baix consum de Microchip.*

7. BIBLIOGRAFIA I RECURSOS WEB

Llibres:

Física per a estudiants d'informàtica, Antoni Giró Roca, UOC , 2005

Sistema per la gestió de fonts d'energies renovables, Miquel Mariño Espinosa, 2009

Manuals electrònics de consulta:

nanoWatt XLP eXtreme Low Power PIC® MCUs, Microchip Inc, http://ww1.microchip.com/downloads/en/DeviceDoc/39941d.pdf

High-Speed Bootloader for PIC16 and PIC18 Devices, Microchip Inc, http://ww1.microchip.com/downloads/en/AppNotes/01310a.pdf

Datasheets:

PIC16F/LF1826/27 Data Sheet, Microchip Inc, http://ww1.microchip.com/downloads/en/DeviceDoc/41391C.pdf

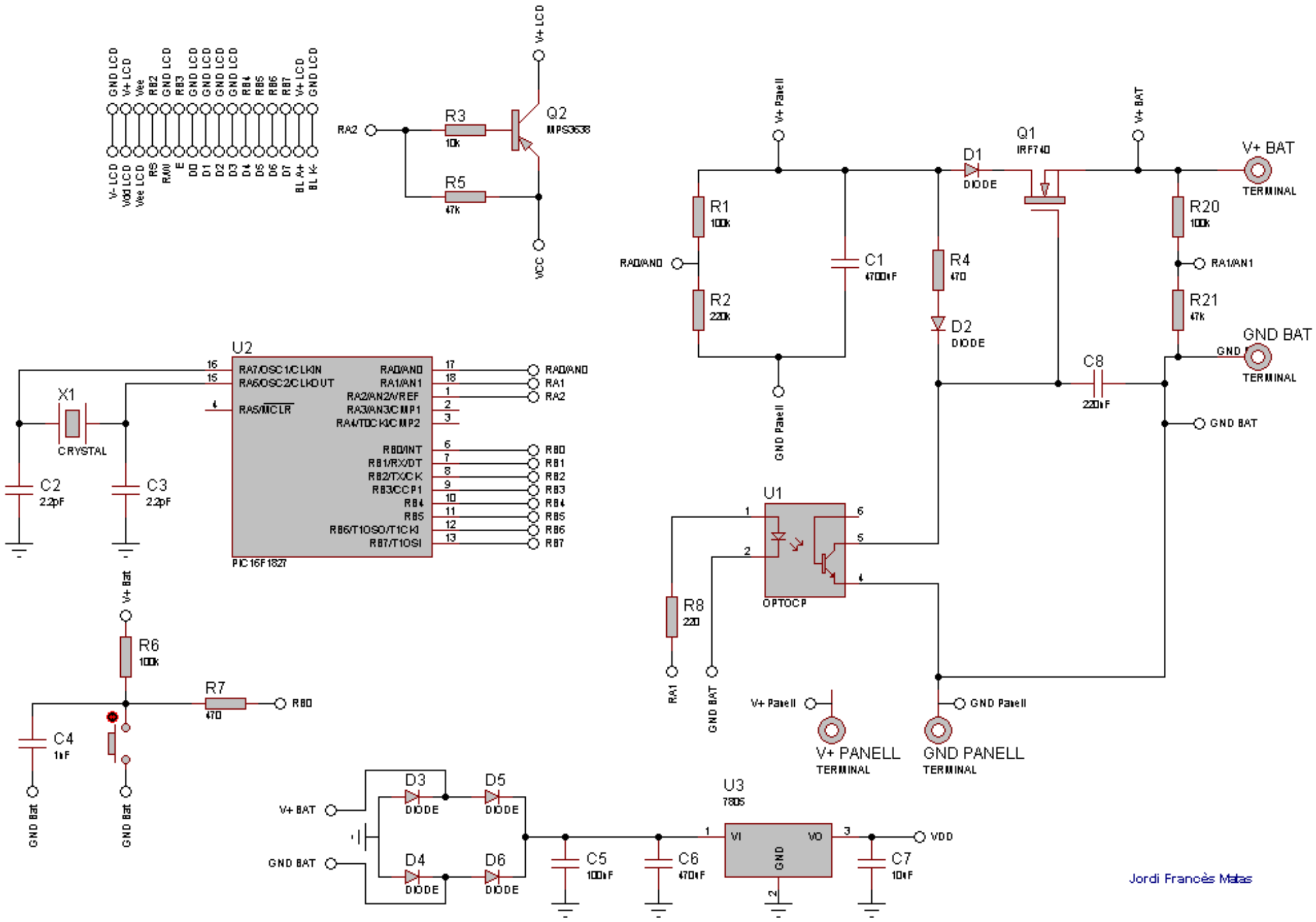
KS006 Datasheet, Samsung, http://www.samsung.com/Products/Semiconductor/

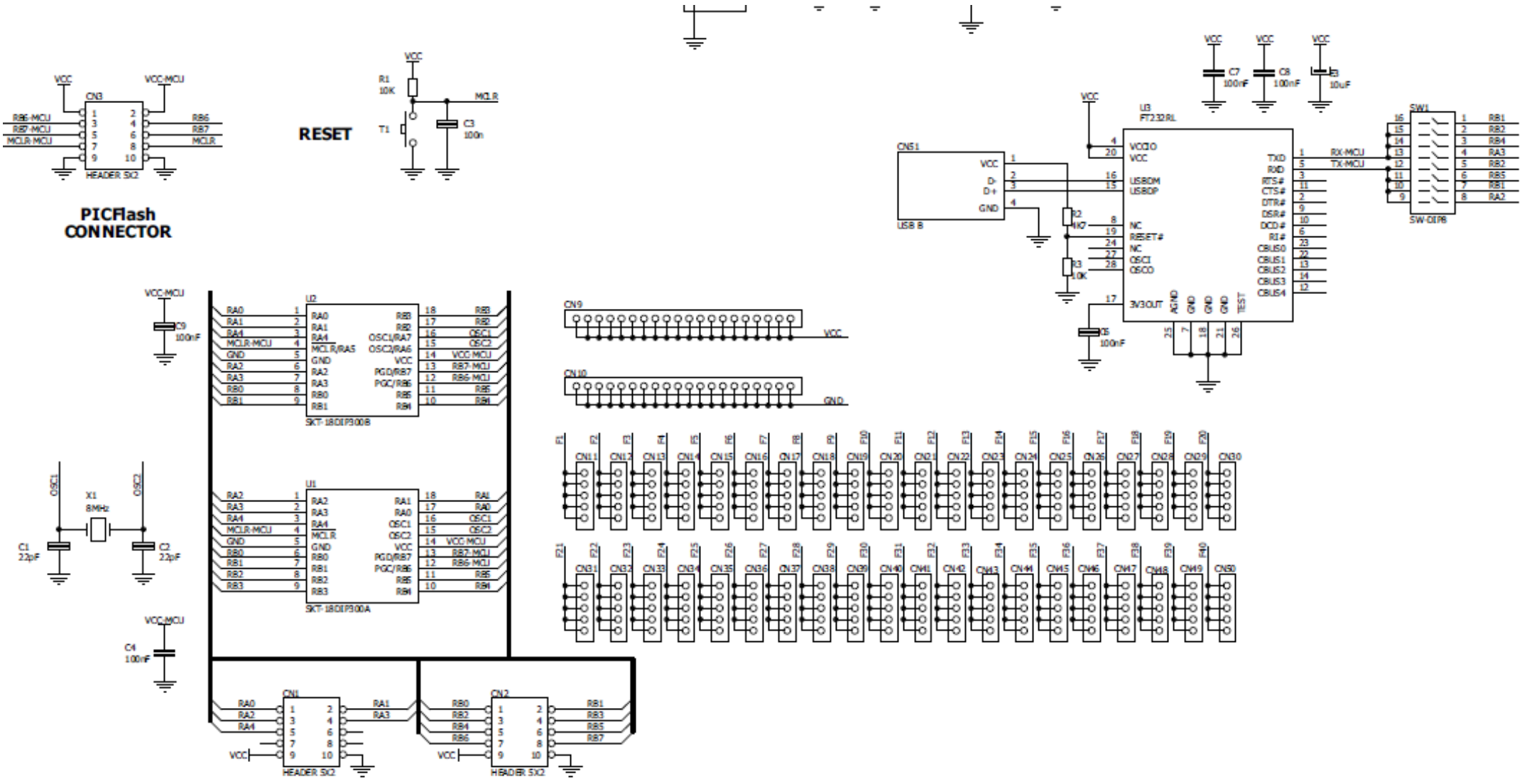
Llocs web:

Microchip, Inc www.microchip.com

Mikroelektronica www.mikroe.com

8. ANNEX A: ESQUEMES ELÈCTRICS


GND LCD
V+ LCD
Vee
RB2
GND LCD
RB3
GND LCD
GND LCD
GND LCD
RB4
RB5
RB6
RB7
V+ LCD
GND LCD
V- LCD
Vdd LCD
Vee LCD
RS
R/W
E
D0
D1
D2
D3
D4
D5
D6
D7
BL A+
BL K-
V+ LCD
R3
10k
Q2
MPS3638
RA2
R5
47k
VCC
U2
RA7/OSC1/CLKIN
RA6/OSC2/CLKOUT
RA5/MCLR
RA0/AN0
RA1/AN1
RA2/AN2/VREF
RA3/AN3/CMP1
RA4/T0CKI/CMP2
RB0/INT
RB1/RX/DT
RB2/TX/CK
RB3/CCP1
RB4
RB5
RB6/T1OSO/T1CKI
RB7/T1OSI
PIC16F1827
X1
CRYSTAL
C2
22pF
C3
22pF
V+ Bat
R6
100k
R7
470
RB0
C4
1nF
GND Bat
GND Bat
V+ Panell
R1
100k
R2
220k
C1
4700nF
GND Panell
D1
DIODE
Q1
IRF740
V+ BAT
V+ BAT
TERMINAL
R20
100k
RA1/AN1
R21
47k
GND BAT
GND BAT
TERMINAL
C8
220nF
GND BAT
R4
470
D2
DIODE
U1
OPTOCP
R8
220
RA1
GND BAT
V+ Panell
GND Panell
V+ PANELL
TERMINAL
GND PANELL
TERMINAL
D3
DIODE
D5
DIODE
D4
DIODE
D6
DIODE
V+ BAT
GND BAT
U3
7805
VI
GND
VO
VDD
C5
100nF
C6
470nF
C7
10nF
Jordi Francès Matas


PICFlash
CONNECTOR
RESET
HEADER SX2
SKT-18DIP300B
SKT-18DIP300A
FT232RL
USB B
SW-DIP8
8MHz

9. ANNEX B: CODIS FONT

UART.h

```
#include <htc.h>
#include "../delay/delay.h"

//Funcions
extern void initUART();
extern void sendUART(char* data,int size);
extern void printUART(const char* s);
extern void printIntUART(int valor);
extern void printPromptUART(const char* prompt,int valor);
```

UART.c

```
#include "uart.h"


void initUART(){
      SPBRGH = 0;
      SPBRGL = 25;
      BRG16 = 0;
    TXSTA = 0b00100100;
    BAUDCON = 0b00000000;
}

void sendUART(char* data,int size){
    char i;
    while(!OSTS){}

    RB3 = 0; //E del LCD a 0
    SPEN = 1;
    for( i = 0; i < size; i++){
        TXREG = data[i];
        while (!TRMT){}
    }
    SPEN = 0;
}
```

```
void printUART(const char* s){
    char i;
    while(!OSTS){}

    RB3 = 0; //E del LCD a 0
    SPEN = 1;
    for( i = 0; s[i] != 0x00; i++){
        TXREG = s[i];
        while (!TRMT){}
    }
    SPEN = 0;
}

void printIntUART(int valor){
    char sortida[6] = {0,0,0,0,0,0};
    char i;
    i = 5;
    while(!OSTS){}
    do {
        sortida[i] = (char) ('0'+ (char)(valor%10));
        valor=valor/10;
        i--;
    }  while (valor > 0);
    i++; //Desfer volta extra

    RB3 = 0; //E del LCD a 0
    SPEN = 1;
    for (; i < 6 ; i++) {
        TXREG = sortida[i];
        while(!TRMT){}
    }
    SPEN=0;
}
```

```
void printPromptUART(const char* prompt,int valor){
    printUART(prompt);
    printIntUART(valor);
    printUART("\r\n");
}
```

ADC.h

```
#include <htc.h>

#define DAC      0b00011110
#define FVR_IN   0b00011111
#define VSS      0b00000000
#define VREF_NEG 0b00000001
#define VDD      0b00000000
#define VREF_POS 0b00000010
#define FVR_REF  0b00000011

extern  void  readADC(char  channel,char  referencia_pos,  char
referencia_neg);
extern void initDAC(char valor);
extern void initFVR(char valorDAC,char valorAD);
extern bit id_resultat;
```

ADC.C

#include "adc.h"

```
#include "../delay/delay.h"
#include "../UART/uart.h"


void initDAC(char valor){
    DACCON1 = valor;
    DACCON0 = 0b11000000;
}
```

```
void initFVR(char valorDAC,char valorAD){
    FVRCON = 0b10000000 | valorDAC << 2 | valorAD;
    espera_ms(2);
}

void readADC(char channel,char referencia_pos, char
referencia_neg){
    ADCON1 = 0b11110000 | (referencia_neg << 2) |
referencia_pos;
    ADCON0 = (channel << 2) | 0x01; //Configurar el canal i
encendre AD
    ADIF=0;
    ADIE=1;
    espera_ms(3); //carrega del Adq. Capacitor
    ADGO = 1;
    SLEEP(); //Dormir durant la conversió
    NOP();
}
```

delay.h

```
#define FREQ 8000 //Definir freq en Hz

//funcions
void espera_ms(unsigned int ms);
```

delay.c

```
#include "delay.h"

void espera(char _ms)
{
  while (--_ms)
  {
  _asm
   nop
_endasm;
      }
}
void espera_ms(unsigned int ms)
{
     ms = ms * (FREQ / 4000);
    while (ms > 0 )
    {
        espera(242);
        ms--;
    }
}
```

LCD.h

```
#include <htc.h>

#define E  RB3
#define RS RB2
#define RW RA4

//Constants per a lcdCommand
#define CLEAR 0x00
#define HOME  0x02

extern char isBusy();
extern void initLCD();
extern void lcdSet4BitMode();
extern void lcdPutc(char c);
```

```
extern void lcdCommand(char c);
extern void lcdSetDDRAM(char c);
extern void lcdGotoXY(char x, char y);
extern void lcdPrint(char* s);
extern void lcdPrintConst(const char* s);
extern void lcdPrintInt(int valor,int pos_x,int pos_y);
extern  void  lcdPrintLong(unsigned  long  valor,int  pos_x,int
pos_y);
```

LCD.c

```
#include "lcd.h"
#include "../delay/delay.h"



void send4bits(char c,char rs){

     PORTB = c & 0xF0;
     RS = rs;
     E = 1;
     espera_ms(1);
     E=0;

     PORTB = (c & 0x0F)<<4;
     RS = rs;
     E = 1;
     espera_ms(1);
     E=0;
     espera_ms(3);
}
```

```
void initLCD(){


     espera_ms(45); //Esperar mes de 30 ms

     PORTB = PORTB | (0b0010<<4);
     RS = 0;
     E = 1;
     espera_ms(1);
     E=0;

     PORTB = PORTB | (0b0010<<4);
     RS = 0;
     E = 1;
     espera_ms(1);
     E=0;

     PORTB = PORTB | (0b1100<<4);
     RS = 0;
     E = 1;
     espera_ms(1);
     E=0;

     espera_ms(2); //Espera mes de 39 us
     send4bits(0b00001100,0);
     espera_ms(2); //Espera mes de 39 us
     send4bits(0b00000001,0);
     espera_ms(3); //Espera mes de 1.53 ms
     send4bits(0b00000110,0);
     espera_ms(5);  //deixem  l'LCD  lliure  per  a  futures
peticions
}

void lcdPutc(char c){
    send4bits(c,1);
    espera_ms(3); //Deixem el display lliure per a futures
interaccions
}
```

```
void lcdCommand(char c){
    send4bits(c,0);
    espera_ms(3); //Deixem el display lliure per a futures
interaccions
}

void lcdSetDDRAM(char c){
    send4bits(c | 0x80,0);
    espera_ms(3); //Deixem el display lliure per a futures
interaccions
}

void lcdGotoXY(char x, char y){
    if (y == 0) lcdSetDDRAM(0x00+x);
    else if (y == 1) lcdSetDDRAM(0x40+x);
    else if (y == 2) lcdSetDDRAM(0x14+x);
    else lcdSetDDRAM(0x54+x);
}

void lcdPrint(char* s){
    char i = 0;
    while(s[i] != 0x00){
        lcdPutc(s[i]);
        i++;
    }

}

void lcdPrintInt(int valor,int pos_x,int pos_y){
       lcdGotoXY(pos_x,pos_y);
       do{
               lcdPutc('0'+(valor%10));
               --pos_x;
               lcdGotoXY(pos_x,pos_y);
               valor/=10;
       }while( valor != 0);
       lcdPutc(' ');
}
```

```
void lcdPrintLong(unsigned long valor,int pos_x,int pos_y){
       lcdGotoXY(pos_x,pos_y);
       do{
               lcdPutc('0'+(valor%10));
               --pos_x;
               lcdGotoXY(pos_x,pos_y);
               valor/=10;
       }while( valor != 0);
       lcdPutc(' ');
}

void lcdPrintConst(const char* s){
    char i = 0;
    while(s[i] != 0x00){
        lcdPutc(s[i]);
        i++;
    }
}
```

main.c

```
#include <htc.h>
#include "../LCD/lcd.h"
#include "../ADC/adc.h"
#include "../UART/uart.h"
#include "../delay/delay.h"


/*Connexions
PORTA:
    RA4      : Unused
    RA3      : Anode del LED del TIL113
    RA2      : Base del BD-PNP
    RA1      : Divisor de tensió bateria
    RA0      : Divisor de tensió condensador
PORTB:
    RB7..RB4 : Data LCD
    RB3      : E LCD
```

```
    RB2      : RS LCD/UART TX
    RB1      : UART RX
    RB0      : Switch INT
*/


//Constants numeriques
const unsigned long cincmil = 5000;
const unsigned long quatremilnorantasis = 4096;
const unsigned long milvintitres = 1023;
const unsigned long factor_bateria = 3127;
const unsigned long factor_condensador = 5545;
const unsigned long mil = 1000;

//Defines per les estadistiques
#define MIDA_TAULA 10

//Defines per als canals del ADC
#define CONDENSADOR 0
#define BATERIA 1

//Defines per al control de transistors
#define BASE_BD440 RA2
#define GATE_IRF740 RA3


//Variables globals
char estat = 0;
char statsCounter = 0;



bit lcdOn;
bit printar_pantalla;


int resultatAD = 0;
int maxCondensador = 0;
int minCondensador = 0;
```

```
int descarregues = 0;
int carregaBateria = 0;
int carregaCond = 0;
int comptadorCicles = 0;
int medianaCondensador=0;
int medianaBateria=0;
unsigned int timeout_carrega = 5000;
unsigned int timeout_descarrega = 100;
unsigned long mvBateria = 0;
unsigned long mvCondensador = 0;

unsigned int descarregues_totals = 0;
unsigned long timeout_pantalla = 2000;

int dadesCondensador[MIDA_TAULA];
int  dadesBateria[MIDA_TAULA];
int  dadesBateriaDetall[MIDA_TAULA];


void interrupt rsi(){
    if (INTF && INTE){
        lcdOn = lcdOn^1;
        printar_pantalla = lcdOn;
        BASE_BD440 = BASE_BD440^1;
        timeout_pantalla = 2000;
        INTF = 0;
    } if (ADIF){
        ADON = 0;
        resultatAD = ADRESH<<8 | ADRESL;
        ADIF = 0;
    }
}

void clearTables(){
    char i;
    for(i = 0; i < MIDA_TAULA; i++){
        dadesCondensador[i] = -1;
        dadesBateria[i] = -1;
        dadesBateriaDetall[i] = -1;
```

```
    }
}

void insertSort(int value,char table){
    int* taula;
    int aux;
    char i;
    char done = 0;
    if (table == CONDENSADOR) taula=dadesCondensador;
    else if(table == BATERIA) taula=dadesBateria;
    else taula=dadesBateriaDetall;
    for(i = 0; i < MIDA_TAULA && !done; i++){
        if(taula[i] == -1){
            taula[i]=value;
            done = 1;
        } else if (taula[i] > value) {
            aux=taula[i];
            taula[i]=value;
            value=aux;
        }
    }



}

void init(){
    //Configuració dels ports
    PORTA  =  0;
    TRISA  =  0b00000011;
    ANSELA =  0b00000011;

    PORTB  =  0;
    TRISB  =  0b00000011;
    ANSELB =  0b00000000;

    //Inicialitzar taules estadistiques
     clearTables();
```

```
    //Estat Inicial dels transistors
    BASE_BD440 = 1;  //LCD OFF
    GATE_IRF740 = 1; //Bateria aillada dels condensadors
    //Inicialització dels periferics
    initUART();
    initFVR(0,3); //Tenir una referencia a 4.096 V per a l'AD

    //Interrupcions
    INTEDG = 0; //Interrupcions en el flanc descendent a pin
INt (RB0)
    ADIE = 1;
    INTCON = 0b11010000;

    //Lògica del programa
    lcdOn = 0;
    estat = 0;
    comptadorCicles = 0;
}

void calibrar(){
    INTE=0;
    GATE_IRF740 = 1; //Bateria aillada dels condensadors
    WDTCON = 0b00011001; //Programar el WDT a 4s i adormir el
PIC
    SLEEP();
    SWDTEN = 0; //Aturar el WDT
    readADC(CONDENSADOR,VDD,VSS);
    maxCondensador = resultatAD - 5; //-5 per la càrrega
asimptotica

    GATE_IRF740 = 0;
    espera_ms(200);
    espera_ms(200);
    readADC(CONDENSADOR,VDD,VSS);
    minCondensador = resultatAD + 5; //+5 per evitar la
descarrega asimptotica
    GATE_IRF740 = 1;

    readADC(BATERIA,VDD,VSS);
```

```
    carregaBateria = resultatAD; //-5 per la càrrega
asimptotica

    timeout_carrega = 5000;
    timeout_descarrega = 100;
    INTF=0;
    INTE=1;
    estat=1;
}

void esperarCarrega(){
    timeout_carrega--;
    if(timeout_carrega==0) estat=0;
    readADC(CONDENSADOR,VDD,VSS);
    while(ADGO);
    if(resultatAD >= maxCondensador){
        estat = 2;
        carregaCond=resultatAD;
        timeout_carrega=5000;
    }
}

void descarrega(){
    timeout_descarrega--;
    if(timeout_descarrega==0) estat=0;
    GATE_IRF740 = 0;
    readADC(CONDENSADOR,VDD,VSS);
    while(ADGO);
    if(resultatAD <= minCondensador){
        estat = 3;
        carregaCond=resultatAD;
        ++descarregues;
        ++descarregues_totals;
        GATE_IRF740 = 1;
        timeout_descarrega=100;
    }

}
```

```
void estadistiques(){
    insertSort(carregaCond,CONDENSADOR);
    readADC(BATERIA,VDD,VSS);        //821  ==  4,0127  V@AD  ~
4,096V@AD
    while(ADGO);
    insertSort(resultatAD,BATERIA);
    if (resultatAD < 821){
        readADC(BATERIA,FVR_REF,VSS);     //Afinar la lectura
per a valors baixos
        while(ADGO);
        insertSort(resultatAD,3);
    } else insertSort(1023,3);
    statsCounter++;
    if(statsCounter >= MIDA_TAULA){
        estat = 4;
        statsCounter = 0;
    }  else estat=1;
}

void informar(){
    while(!OSTS);
    printPromptUART("Condensador:
",dadesCondensador[MIDA_TAULA/2]);
    printPromptUART("Bateria
(5V)",dadesBateria[MIDA_TAULA/2]);
    printPromptUART("Bateria
(4.096V)",dadesBateriaDetall[MIDA_TAULA/2]);
    clearTables();
    estat=1;
}

void lcd_usage(){
if (lcdOn){
            if(printar_pantalla){
                initLCD();
                lcdPrintConst("B: ");
                mvBateria                                    =
((((carregaBateria*cincmil)/milvintitres)*factor_bateria)/mil
)+900; //+900 per corregir l'offset dels diodes de la placa
```

```
                lcdPrintLong(mvBateria,6,0);
                lcdGotoXY(7,0);
                lcdPrintConst("mV D: ");
                lcdPrintInt(descarregues_totals,15,0);

                lcdGotoXY(0,1);
                lcdPrintConst("C: ");
                mvCondensador                                   =
((((maxCondensador*cincmil)/milvintitres)*factor_condensador)
/mil)+900; //+900 per corregir l'offset dels diodes de la
placa
                lcdPrintLong(mvCondensador,7,1);
                lcdGotoXY(8,1);
                lcdPrintConst("mV");
                PORTB=0x00;
                printar_pantalla=0;
            }
            timeout_pantalla--;
            if(timeout_pantalla == 0) {
                timeout_pantalla = 2000;
                lcdOn=0;
                BASE_BD440 = 1;
            }
        }
}
```

```
void main(){
    init();
    while(1){
        lcd_usage();
        if (estat == 0) calibrar();
        if (estat == 1) esperarCarrega();
        if (estat == 2) descarrega();
        if (estat == 3) estadistiques();
        if (estat == 4) informar();

        if (descarregues >= 200 && estat==1){
            descarregues = 0;
            estat=0;
        }
    }
}
```

Bootloader

```
#include "devices.inc"
#include "bankswitch.inc"
#include "bootconfig.inc"
;
************************************************************
***************

;
************************************************************
***************

#if BOOTLOADERSIZE < ERASE_FLASH_BLOCKSIZE
    ; This device has a large Erase FLASH Block Size, so we
need to reserve a full Erase Block
    ; page for the bootloader. Reserving an entire erase
block prevents the PC application
    ; from accidently erasing a portion of the bootloader.
    #define BOOTBLOCKSIZE ERASE_FLASH_BLOCKSIZE
```

```
    #ifndef BOOTLOADER_ADDRESS
        #ifdef CONFIG_AS_FLASH
            #define   BOOTLOADER_ADDRESS      (END_FLASH    -
BOOTBLOCKSIZE - ERASE_FLASH_BLOCKSIZE)
        #else
            #define   BOOTLOADER_ADDRESS      (END_FLASH    -
BOOTBLOCKSIZE)
        #endif
    #endif
#else
    #if (BOOTLOADERSIZE % ERASE_FLASH_BLOCKSIZE) == 0
        #define BOOTBLOCKSIZE BOOTLOADERSIZE
    #else
        #define      BOOTBLOCKSIZE      (BOOTLOADERSIZE      /
ERASE_FLASH_BLOCKSIZE + 1) * ERASE_FLASH_BLOCKSIZE
    #endif
    #ifndef BOOTLOADER_ADDRESS
        #define    BOOTLOADER_ADDRESS        (END_FLASH    -
BOOTBLOCKSIZE)
    #endif
#endif

#ifndef AppVector
    ;  The  application  startup  GOTO  instructions  will  be
written just before the Boot Block,
    ; courtesy of the host PC bootloader application.
    #define AppVector (BootloaderStart-.5)
#endif

;
*************************************************************
***************
#define STX             0x0F
#define ETX             0x04
#define DLE             0x05
#define NTX             0xFF
;
*************************************************************
***************
```

```
;
*****************************************************************
***************
CRCL                equ 0xA0        ; GPR RAM in bank 1
CRCH                equ 0xA1
RXDATA              equ 0xA2
TXDATA              equ 0xA3

; Framed Packet Format
;
<STX>[<COMMAND><ADDRL><ADDRH><ADDRU><0x00><DATALEN><...DATA..
.>]<CRCL><CRCH><ETX>

COMMAND             equ 0x20        ; receive buffer in bank
0
ADDRESS_L           equ 0x21
ADDRESS_H           equ 0x22
ADDRESS_U           equ 0x23
ADDRESS_X           equ 0x24
DATA_COUNTL         equ 0x25
PACKET_DATA         equ 0x26
DATA_COUNTH            equ 0x26          ; only for certain
commands

#if BOOTLOADER_ADDRESS == 0
#ifndef BSR
PCLATH_TEMP         equ 0x7E        ;    Interrupt    context
save/restore temporary memory
W_TEMP                  equ 0x7F
#endif
#endif
;
*****************************************************************
***************

#ifdef PORTC
    ; Most parts have the UART on RC7 and RC6 pins.
    #define PORTRX  PORTC
```

```
    #define RXPIN   7

    #define TRISTX  TRISC
    #define TXPIN   6
#else
    ;  PIC16F88  doesn't  have  PORTC,  so  the  UART  pins  are
elsewhere...
    #define PORTRX  PORTB
    #define RXPIN   1

    #define TRISTX  TRISB
    #define TXPIN   2
#endif

    errorlevel -302                 ; Do not show any banking
warnings
;
************************************************************
***************
#if BOOTLOADER_ADDRESS != 0
    ORG     0
    ; The following GOTO is not strictly necessary, but may
startup faster
    ; if running at slow clock speeds.
        errorlevel -306               ; Do not show any page
boundary warnings
    ;nop
    ;movlw   high(BootloaderBreakCheck)
    ;movwf   PCLATH                  ; Bx
    movlp   high(BootloaderBreakCheck)
    goto    BootloaderBreakCheck
        errorlevel +306               ; Do not show any page
boundary warnings

    ORG     BOOTLOADER_ADDRESS
BootloaderStart:
    movlw   high(BootloadMode)
    movwf   PCLATH                  ; Bx
    goto    BootloadMode
```

```
;
*************************************************************
***************
; Determine if the application is supposed to be started or
if we should
; go into bootloader mode.
;
; If RXD is in BREAK state (vs IDLE) when we come out of MCLR
reset,
; immediately enter bootloader mode, even if there exists
some application
; firmware in program memory.
BootloaderBreakCheck:
    movlw   high(AppVector)
    movwf   PCLATH                  ; Bx
#ifdef INVERT_UART
    btfss   PORTRX, RXPIN           ; B0
    goto    AppVector               ; no BREAK state, attempt
to start application

    btfsc    PORTRX, RXPIN            ; B0 BREAK found, wait
for RXD to go IDLE
    goto    $-1
#else
    banksel ANSELB
    bcf ANSELB, RXPIN
    banksel TRISB
    bsf TRISB, RXPIN                ;Jordi: defining RXPIN as
a digital input
    banksel PORTB


    btfsc   PORTRX, RXPIN           ; B0
    goto    AppVector               ; no BREAK state, attempt
to start application

    btfss    PORTRX, RXPIN            ; B0 BREAK found, wait
for RXD to go IDLE
```

```
    goto    $-1

#endif
#else         ;         BOOTLOADER_ADDRESS         ==         0
*************************************************************
***
    ORG     0
BootloaderStart:
    nop                             ; required to allow debug
executive startup when running under ICD
    BXtoBO                          ; Bx -> BO
    goto    BootloaderBreakCheck

    ORG     0x0004
InterruptVector:
#ifndef BSR
      movwf W_TEMP                      ; Bx save W register
temporarily
      swapf PCLATH, W                    ;  Bx  save  PCLATH
register
      movwf PCLATH_TEMP                  ; Bx (SWAPF used to
avoid damaging STATUS register)
#endif
    movlw    high(AppIntVector)         ; Bx set PCLATH for
making a long jump to the AppIntVector address
    movwf   PCLATH                  ; Bx
    goto     AppIntVector              ; Bx jump to remapped
application interrupt vector.

BootloaderBreakCheck:
#ifdef INVERT_UART
    btfsc   PORTRX, RXPIN           ; BO
    goto    WaitForRxIdle           ; BREAK detected, startup
in Bootloader mode
#else
    btfss   PORTRX, RXPIN           ; BO
    goto    WaitForRxIdle           ; BREAK detected, startup
in Bootloader mode
#endif
```

```
    ; Attempt to startup in Application mode.
    ;  Read  instruction  at  the  application  reset  vector
location.
    ; If we read 0x3FFF, assume that the application firmware
has
    ;  not  been  programmed  yet,  so  don't  try  going  into
application mode.
    banksel EEADR                   ; Bx -> B2
    movlw    low(AppVector)             ; Bx load address of
application reset vector
    movwf   EEADR                   ; B2
    movlw   high(AppVector)
    movwf   EEADRH                  ; B2
    movwf   PCLATH                  ; Bx
    call    ReadFlashWord           ; Bx -> B0

    addlw   .1
    btfss   STATUS, Z               ; Bx if the lower byte !=
0xFF,
    goto    AppVector               ; Bx run application.

    movlw   0x3F
    xorwf   FSR, w                  ; Bx if the lower byte ==
0xFF but upper byte != 0x3F,
    btfss   STATUS, Z               ; Bx run application
    goto    AppVector

    movlw   high(BootloadMode)
    movwf   PCLATH                  ; Bx

    ; otherwise, assume application firmware is not loaded,
    ; fall through to bootloader mode...
#ifdef INVERT_UART
WaitForRxIdle:
    btfsc    PORTRX, RXPIN             ; B0 BREAK found, wait
for RXD to go IDLE
    goto    WaitForRxIdle
#else
```

```
WaitForRxIdle:
    btfss    PORTRX, RXPIN            ; B0 BREAK found, wait
for RXD to go IDLE
    goto    WaitForRxIdle
#endif
#endif      ;      end      BOOTLOADER_ADDRESS      ==      0
****************************************

BootloadMode:

    banksel ANSELB
    clrf ANSELB

#ifdef BRG16
    movlw    b'00110000'                ; 1:8 prescaler - no
division required later (but no rounding possible)
    movwf   T1CON                   ; B0
#endif

#ifdef BSR
    banksel RCSTA
    movlw   b'10010000'             ; Setup UART
    movwf   RCSTA                   ; B0

    bcf      TRISTX, TXPIN             ; B1 Setup TX pin for
output
    movlw   b'00100110'             ; BRGH = 1, TXEN = 1
    movwf   TXSTA                   ; B1
    banksel OPTION_REG
#else
    movlw   b'10010000'             ; Setup UART
    movwf   RCSTA                   ; B0

    B0toB1                          ; B0 -> B1
    bcf      TRISTX, TXPIN             ; B1 Setup TX pin for
output
    movlw   b'00100110'             ; BRGH = 1, TXEN = 1
    movwf   TXSTA                   ; B1
#endif
```

```
#ifndef BRG16
    clrwdt                          ; required to avoid reset
when modifying TMR0 prescaler assignment
    movlw    b'00000011'                ; 1:16 prescaler for
Timer 0, used for auto-baud calculation
    movwf   OPTION_REG              ; B1
#endif

    B1toB3                          ; B1 -> B3
#ifdef INVERT_UART
    bsf     BAUDCTL, RXDTP          ; B3
    bsf     BAUDCTL, TXCKP          ; B3
#endif
#ifdef BRG16
    bsf     BAUDCTL, BRG16          ; B3
#endif
;
*****************************************************************
****************

;
*****************************************************************
****************
DoAutoBaud:
; ___    __________            ________
;    \__/          \__________/
;       |                      |
;       |-------- p ----------|
;
;   p = The number of instructions between the first and last
;            rising edge of the RS232 control sequence 0x0F.
Other
;         possible control sequences are 0x01, 0x03, 0x07,
0x1F,
;       0x3F, 0x7F.
;
;   SPBRG = (p / 32) - 1    BRGH = 1, BRG16 = 0
;   SPBRG = (p / 8) - 1     BRGH = 1, BRG16 = 1
```

```
   ;Possible fixed baudrates, assuming 8 Mhz Fosc, table 25-5
on datasheet page 302

   ;19.2k
   banksel SPBRGL
   movlw .25
   movwf SPBRGL
   clrf SPBRGH
   bcf BAUDCON, BRG16
   ;bcf TXSTA, SYNC
   ;bsf TXSTA, BRGH                    ;Jordi: Unnecessary to
modify, since it's been done by Microchip's code

   ;9600 mode 1
   ;banksel SPBRGL
   ;movlw .51
   ;movwf SPBRGL
   ;clrf SPBRGH
   ;bcf BAUDCON, BRG16
   ;bcf TXSTA, SYNC
   ;bsf TXSTA, BRGH

   ;9600 mode 2
   ;banksel SPBRGL
   ;movlw .12
   ;movwf SPBRGL
   ;clrf SPBRGH
   ;bcf BAUDCON, BRG16
   ;bcf TXSTA, SYNC
   ;bcf TXSTA, BRGH

   bcf BAUDCON,ABDEN; disable Auto-Baudrate

WaitForHostCommand:                ; B0/B1
#ifdef BSR
    banksel RCSTA
#else
```

```
    B1toB0                          ; B1 -> B0
#endif
    bsf     RCSTA, CREN             ; B0 start receiving

    lfsr    COMMAND                 ; Bx Point to the buffer
    call      ReadHostByte                ; B0 get start of
transmission <STX>
    xorlw   STX
    bnz       DoAutoBaud                ; Bx got something
unexpected, perform autobaud
;
*************************************************************
***************

;
*************************************************************
***************
; Read and parse packet data.
StartOfLine:
    movlw    STX                        ; send back start of
response
    call    SendHostByte            ; B0/B1 -> B1

ReceiveDataLoop:
    call    ReadHostByte            ; Bx -> B0 Get the data
    xorlw   STX                     ; Check for an unexpected
STX
    bz       StartOfLine              ; unexpected STX: abort
packet and start over.

NoSTX:
    movf    INDF, W                 ; Bx
    xorlw   ETX                     ; Check for a ETX
    bz      VerifyPacketCRC         ; Yes, verify CRC

NoETX:
    movf    INDF, W                 ; Bx
    xorlw   DLE                     ; Check for a DLE
    bnz     AppendDataBuffer
```

```
    call    ReadHostByte            ; Bx -> BO DLE received,
get the next byte and store it

AppendDataBuffer:
    incf    FSR, f                  ; Bx move to next empty
location
    goto    ReceiveDataLoop

VerifyPacketCRC:
    BOtoB1                          ; BO -> B1
    decf    FSR, w                  ; Bx
    movwf   TXDATA                  ; B1 save end of packet
pointer
    decf    TXDATA, f               ; Bx

    lfsr    COMMAND                 ; Bx reset pointer to
beginning of data
    clrf    CRCL                            ; B1 reset CRC
accumulator
    clrf    CRCH                    ; B1

VerifyPacketCrcLoop:
    movf    INDF, w                 ; Bx
    call    AddCrcB1                ; B1 add new data to the
CRC

    incf    FSR, f                  ; Bx
    movf    FSR, w                  ; Bx
    subwf   TXDATA, w               ; B1
    bnz     VerifyPacketCrcLoop     ; we aren't at the end of
the received data yet, loop

    movf    CRCL, w                 ; B1
    subwf   INDF, w                 ; Bx
    bnz     DoAutoBaud              ; invalid CRC, reset baud
rate generator to re-sync with host

    incf    FSR, f                  ; Bx
```

```
    movf    CRCH, w                 ; B1
    subwf   INDF, w                 ; Bx
    bnz     DoAutoBaud                ; Bx invalid CRC, reset
baud rate generator to re-sync with host

; ********************************************
; Pre-setup, common to all commands.
    clrf    CRCL                    ; B1
    clrf    CRCH                    ; B1

    B1toB0                          ; B1 -> B0
    movf    ADDRESS_H, W            ; B0 read address pointer
from packet data
    movwf    FSR                      ; Bx temporarily save
high address byte to FSR
    movf    ADDRESS_L, W            ; B0
    banksel EEADR                   ; Bx -> B2
    movwf   EEADR                   ; B2
    movf     FSR, w                     ; Bx read back high
address byte from temporary register
    movwf   EEADRH                  ; B2

    lfsr    PACKET_DATA             ; Bx
    BXtoB0                          ; Bx -> B0

; ********************************************



; ********************************************
; Test the command field and sub-command.
CheckCommand:
    movlw   (JUMPTABLE_END - JUMPTABLE_BEGIN)
    subwf    COMMAND, w                 ; B0 test for valid
command number
    bc       DoAutoBaud               ; Bx invalid command -
reset baud generator and re-sync with host

    movf    COMMAND, W              ; B0
```

```
    ; This jump table must exist entirely within one 256 byte
block of program memory.
#if ($ & 0xFF) > (0xFF - .10)
    ; Too close to the end of a 256 byte boundary, push
address forward to get code
    ; into the next 256 byte block.
    messg   "Wasting some code space to ensure jump table is
aligned."
    ORG     $+(0x100 - ($ & 0xFF))
#endif
    addwf   PCL, F                    ; 0 Bx Jump in command
jump table based on COMMAND from host
JUMPTABLE_BEGIN:
    goto    BootloaderInfo          ; 1 B0 0
    goto    ReadFlash               ; 2 Bx 1
    goto    VerifyFlash             ; 3 Bx 2
    goto    EraseFlash              ; 4 Bx 3
    goto    WriteFlash              ; 5 Bx 4
    goto    ReadEeprom              ; 6 B0 5
    goto    WriteEeprom             ; 7 Bx 6
    goto     SendAcknowledge          ; 8 B0 7 - WriteConfig
not supported on PIC16F devices
    nop                             ; 9 B0 8
    movlw   high(AppVector)         ; 10 B0 9
JUMPTABLE_END:
    movwf   PCLATH                  ; Bx
    goto    AppVector               ; B0
#if (JUMPTABLE_BEGIN & 0xFF) > (JUMPTABLE_END & 0xFF)
    error "Jump table is not aligned to fit within a single
256 byte address range."
#endif

WaitForRise:                        ; B0
    clrwdt

WaitForRiseLoop:                    ; B0
#ifdef BRG16
    btfsc   PIR1, TMR1IF             ; B0 if TMR1 overflowed,
we did not get a good baud capture
```

```
    return                          ; abort
#endif

    btfsc   PORTRX, RXPIN           ; B0 Wait for a falling
edge
    goto    WaitForRiseLoop         ; B0

WtSR:
    btfss   PORTRX, RXPIN           ; B0 Wait for rising edge
    goto    WtSR                    ; B0
    return

; 16-bit CCIT CRC
; Adds WREG byte to the CRC checksum CRCH:CRCL. WREG
destroyed on return.
AddCrc:                             ; B0/B1 Init: CRCH = HHHH
hhhh, CRCL = LLLL llll
    B0toB1                          ; B0 -> B1
AddCrcB1:
    xorwf   CRCH, w                     ; B1 Pre:  HHHH hhhh
WREG =      IIII iiii
    movwf   RXDATA                  ; B1
    movf    CRCL, w                     ; B1 Pre:  LLLL llll
CRCH =      LLLL llll
    movwf   CRCH                    ; B1
    movf    RXDATA, w               ; B1
    movwf   CRCL                        ; B1 Pre:  IIII iiii
CRCL =      IIII iiii
    swapf   CRCL, w                     ; B1 Pre:  IIII iiii
WREG =      iiii IIII
    andlw   0x0F                        ; Pre:  iiii IIII
WREG =      0000 IIII
    xorwf   CRCL, f                     ; B1 Pre:  IIII iiii
CRCL =      IIII jjjj
    swapf   CRCL, w                     ; B1 Pre:  IIII jjjj
WREG =      jjjj IIII
    andlw   0xF0                        ; Pre:  jjjj IIII
WREG =      jjjj 0000
```

```
    xorwf    CRCH, f                    ; B1 Pre:   LLLL llll
CRCH =      MMMM llll
    swapf    CRCL, f                    ; B1 Pre:   IIII jjjj
WREG =      jjjj IIII
    bcf      STATUS, C              ; Bx
    rlf       CRCL, w                   ; B1 Pre:   jjjj IIII
WREG =      jjjI IIIj
    btfsc    STATUS, C              ; Bx
    addlw    .1
    xorwf    CRCH, f                    ; B1 Pre:   MMMM llll
CRCH =      XXXN mmmm
    andlw     b'11100000'                ; Pre:   jjjI IIIj
WREG =      jjj0 0000
    xorwf    CRCH, f                    ; B1 Pre:   jjj0 0000
CRCH =      MMMN mmmm
    swapf    CRCL, f                    ; B1 Pre:   IIII jjjj
WREG =      jjjj IIII
    xorwf    CRCL, f                    ; B1 Pre:   MMMN mmmm
CRCL =      JJJI jjjj
    return

; ********************************************
; Commands
; ********************************************

; Provides information about the Bootloader to the host PC
software.
BootInfoBlock:
    db      high(BOOTBLOCKSIZE), low(BOOTBLOCKSIZE)
    db      MINOR_VERSION, MAJOR_VERSION
#ifdef FREE
    db       0x02, 0x01                ; family id : command
mask (erase flash command enabled)
#else
    db       0x02, 0x00                ; family id : command
mask (no erase flash command)
#endif
    db      high(BootloaderStart), low(BootloaderStart)
    db      0, upper(BootloaderStart)
```

```
    db      high(DEVICEID), low(DEVICEID)
BootInfoBlockEnd:

; In:   <STX>[<0x00>]<CRCL><CRCH><ETX>
;                                                          Out:
<STX><BOOTBYTESL><BOOTBYTESH><VERL><VERH><STARTBOOTL><STARTBO
OTH><STARTBOOTU><0x00><CRCL><CRCH><ETX>
BootloaderInfo:                     ; B0
    movlw   (BootInfoBlockEnd - BootInfoBlock)
    movwf   DATA_COUNTL             ; B0
    clrf    DATA_COUNTH             ; B0

    banksel EEADR
    movlw   low(BootInfoBlock)
    movwf   EEADR                   ; B2/B3(PIC16F193x)
    movlw   high(BootInfoBlock)     ; B2/B3(PIC16F193x)
    movwf   EEADRH                  ; B2/B3(PIC16F193x)

    ;; fall through to ReadFlash code -- send Bootloader
Information Block from FLASH.

;                                                           In:
<STX>[<0x01><ADDRL><ADDRH><ADDRU><0x00><BYTESL><BYTESH>]<CRCL
><CRCH><ETX>
; Out:  <STX>[<DATA>...]<CRCL><CRCH><ETX>
ReadFlash:                          ; Bx -> B0
    call    ReadFlashWord           ; Bx -> B0
    call    SendEscapeByte          ; Bx -> B1
    call    AddCrcB1                ; B1
    movf      FSR, w                        ; Bx read most
significant bits from tempory memory
    call    SendEscapeByte          ; Bx -> B1
    call    AddCrcB1                ; B1

    banksel EEADR                   ; Bx -> B2
    incf    EEADR, f                ; B2
    btfsc   STATUS, Z               ; Bx
    incf    EEADRH,F                ; B2
```

```
    BXtoB0                          ; Bx -> B0
    movlw   1
    subwf   DATA_COUNTL, f          ; B0
    btfss   STATUS, C
    decf    DATA_COUNTH, f          ; B0

    movf        DATA_COUNTH,  w                   ;  B0  is
DATA_COUNTH:DATA_COUNTL == 0?
    iorwf   DATA_COUNTL, w          ; B0
    bnz      ReadFlash                  ; Bx non-zero, keep
reading more data
    goto    SendChecksum            ; Bx zero, exit read loop
and send end of packet

ReadFlashWord:                      ; Bx -> B0
    banksel EECON1                  ; Bx -> B3
    bsf      EECON1, EEPGD              ; B3 access program
memory instead of eeprom data memory
    bsf      EECON1, RD                 ; B3 initiate read
operation
    nop                                ; B3 sounds like this
instruction slot might be usable, but NOP is safest
    nop                               ; B3 required NOP during
program memory read operation
    banksel EEDATA                  ; Bx -> B2
    movf     EEDATH, w                    ; B2 read most
significant bits of program memory
    movwf   FSR                     ; Bx save it temporarily
    movf     EEDATA, w                   ; B2 read least
significant byte of program memory
    BXtoB0                          ; Bx -> B0
    return

;                                                       In:
<STX>[<0x02><ADDRL><ADDRH><ADDRU><0x00><BLOCKSL><BLOCKSH>]<CR
CL><CRCH><ETX>
; Out:  <STX>[<CRCL1><CRCH1>...<CRCLn><CRCHn>]<ETX>
VerifyFlash:                        ; Bx
    call    ReadFlashWord           ; Bx -> B0
```

```
    call    AddCrc                  ; B1/B0 -> B1
    movf      FSR, w                        ; Bx read most
significant bits from tempory memory
    call    AddCrcB1                ; B1

    banksel EEADR                   ; Bx -> B2
    incf    EEADR, f                ; B2
    btfsc   STATUS, Z               ; Bx
    incf    EEADRH,F                ; B2

    movf    EEADR, w                ; B2
    andlw   (ERASE_FLASH_BLOCKSIZE-1)
    bnz     VerifyFlash             ; Bx

    call    SendCRCWord             ; Bx -> B1

    BXtoB0                          ; Bx -> B0
    movlw   1
    subwf   DATA_COUNTL, f          ; B0
    btfss   STATUS, C               ; Bx
    decf    DATA_COUNTH, f          ; B0

    movf        DATA_COUNTH,  w                   ;  B0  is
DATA_COUNTH:DATA_COUNTL == 0?
    iorwf   DATA_COUNTL, w          ; B0
    bnz       VerifyFlash               ; Bx non-zero, keep
reading more data
    goto    SendETX                 ; Bx zero, exit read loop
and send end of packet

;                                                       In:
<STX>[<0x03><ADDRL><ADDRH><ADDRU><0x00><PAGESL>]<CRCL><CRCH><
ETX>
; Out:  <STX>[<0x03>]<CRCL><CRCH><ETX>
#ifdef FREE
#ifdef USE_SOFTBOOTWP
    goto     BootloaderStart            ; this code -should-
never be executed, but in case of errant
```

```
    goto    BootloaderStart          ; execution or firmware
bugs, may protect against accidental erases.
#endif
EraseFlash:                          ; Bx
#ifdef USE_SOFTBOOTWP
  #define ERASE_ADDRESS_MASK   ( (~(ERASE_FLASH_BLOCKSIZE-1))
& (END_FLASH-1) )

    banksel EEADR                    ; Bx -> B2
  #if high(ERASE_ADDRESS_MASK) != 0xFF
    movlw    high(ERASE_ADDRESS_MASK)      ; force starting
address to land on a FLASH Erase Block boundary
    andwf   EEADRH, f
  #endif
  #if low(ERASE_ADDRESS_MASK) != 0xFF
    movlw    low(ERASE_ADDRESS_MASK)       ; force starting
address to land on a FLASH Erase Block boundary
    andwf   EEADR, f
  #endif

  #if BOOTLOADER_ADDRESS != 0
    ; is the address to erase less than the bootloader
address?
    movlw   high(BOOTLOADER_ADDRESS)
    subwf   EEADRH, w
    movlw   low(BOOTLOADER_ADDRESS)
    btfsc   STATUS, Z
    subwf   EEADR, w
    btfss   STATUS, C
    goto    EraseAddressOkay     ; erasing memory before the
boot block is okay
  #endif

    ; is the address to erase greater than or equal to the
end of the boot block?
    movlw   high(BOOTLOADER_ADDRESS + BOOTBLOCKSIZE)
    subwf   EEADRH, w
    movlw   low(BOOTLOADER_ADDRESS + BOOTBLOCKSIZE)
    btfsc   STATUS, Z
```

```
    subwf   EEADR, w
    btfsc   STATUS, C
    goto     EraseAddressOkay     ; erasing memory after the
boot block is okay

    banksel EECON1              ; Bx -> B3
    clrf     EECON1                ; inhibit writes for this
block
    goto    NextEraseBlock      ; move on to next erase block

    goto    BootloaderStart     ; this code -should- never be
executed, but in case of errant
    goto    BootloaderStart     ; execution or firmware bugs,
may protect against accidental writes.
#endif
EraseAddressOkay:
    banksel EECON1                  ; Bx -> B3
    movlw   b'10010100'             ; Bx setup FLASH erase
    movwf   EECON1                  ; B3
    call    StartWriteB3            ; B3 erase the page

NextEraseBlock:
    ; Decrement address by erase block size
    banksel EEADR                   ; Bx -> B2
#if ERASE_FLASH_BLOCKSIZE >= .256
    movlw   high(ERASE_FLASH_BLOCKSIZE)
    subwf   EEADRH, F               ; B2
#else
    movlw   ERASE_FLASH_BLOCKSIZE
    subwf   EEADR, F                ; B2
    btfss   STATUS, C
    decf    EEADRH, f               ; B2
#endif

    banksel DATA_COUNTL             ; Bx -> B1
    decfsz  DATA_COUNTL, F          ; B1
    goto    EraseFlash
    goto      SendAcknowledge             ; All done, send
acknowledgement packet
```

```
#endif

#ifdef USE_SOFTBOOTWP
    goto     BootloaderStart            ; this code -should-
never be executed, but in case of errant
    goto     BootloaderStart           ; execution or firmware
bugs, may protect against accidental writes.
#endif

;                                                              In:
<STX>[<0x04><ADDRL><ADDRH><ADDRU><0x00><BLOCKSL><DATA>...]<CR
CL><CRCH><ETX>
; Out:  <STX>[<0x04>]<CRCL><CRCH><ETX>
WriteFlash:                          ; Bx
#ifdef USE_SOFTBOOTWP
  #define WRITE_ADDRESS_MASK   ( (~(WRITE_FLASH_BLOCKSIZE-1))
& (END_FLASH-1) )

    banksel EEADR                    ; Bx -> B2
  #if high(WRITE_ADDRESS_MASK) != 0xFF
    movlw     high(WRITE_ADDRESS_MASK)      ; force starting
address to land on a FLASH Write Block boundary
    andwf   EEADRH, f
  #endif
  #if low(WRITE_ADDRESS_MASK) != 0xFF
    movlw     low(WRITE_ADDRESS_MASK)       ; force starting
address to land on a FLASH Write Block boundary
    andwf   EEADR, f
  #endif

  #if BOOTLOADER_ADDRESS != 0
    ; is the address to write less than the bootloader
address?
    movlw   high(BOOTLOADER_ADDRESS)
    subwf   EEADRH, w
    movlw   low(BOOTLOADER_ADDRESS)
    btfsc   STATUS, Z
    subwf   EEADR, w
    btfss   STATUS, C
```

```
    goto     WriteAddressOkay     ; writing before the boot
block is okay
  #endif

    ; is the address to write greater than or equal to the
end of the boot block?
    movlw   high(BOOTLOADER_ADDRESS + BOOTBLOCKSIZE)
    subwf   EEADRH, w
    movlw   low(BOOTLOADER_ADDRESS + BOOTBLOCKSIZE)
    btfsc   STATUS, Z
    subwf   EEADR, w
    btfsc   STATUS, C
    goto     WriteAddressOkay     ; writing after the boot
block is okay

    banksel EECON1              ; Bx -> B3
    clrf     EECON1                ; inhibit writes for this
block
    goto     LoadHoldingRegisters; fake the write so we can
move on to real writes

    goto    BootloaderStart     ; this code -should- never be
executed, but in case of errant
    goto    BootloaderStart     ; execution or firmware bugs,
may protect against accidental writes.
#endif
WriteAddressOkay:
    banksel EECON1                  ; Bx -> B3
    movlw   b'10000100'             ; B3 setup FLASH write
    movwf   EECON1                  ; B3

LoadHoldingRegisters:
    banksel EEDATA                  ; Bx -> B2
    movf     INDF, w                    ; Bx read from buffer
memory
    incf     FSR, f                     ; Bx increment buffer
memory pointer
    movwf    EEDATA                      ; B2 load the least
significant byte holding register
```

```
    movf     INDF, w                    ; Bx read from buffer
memory
    incf     FSR, f                     ; Bx increment buffer
memory pointer
    movwf    EEDATH                       ; B2 load the most
significant byte holding register
    call     StartWrite               ; B3 initiate a write

    banksel EEADR
    incf     EEADR, f                  ; B2/B3 increment FLASH
memory write pointer
    btfsc    STATUS, Z                ; Bx
    incf     EEADRH, f                ; B2/B3

    ; are we at the end of a write block?
    movlw    (WRITE_FLASH_BLOCKSIZE-1)
    andwf    EEADR, w                 ; B2/B3
    bnz      LoadHoldingRegisters     ; Bx

    B2toB0                            ; B2 -> B0
    decfsz   DATA_COUNTL, F             ; B0 finished writing
block, is there any more data to write?
    goto     WriteFlash               ; Bx more data to write,
repeat.
    goto     SendAcknowledge          ; B0 all done, send ACK
packet

;                                                              In:
<STX>[<0x05><ADDRL><ADDRH><0x00><0x00><BYTESL><BYTESH>]<CRCL>
<CRCH><ETX>
; Out:  <STX>[<DATA>...]<CRCL><CRCH><ETX>
ReadEeprom:                           ; Bx
    banksel EECON1                    ; Bx -> B3
    clrf     EECON1                   ; B3

ReadEepromLoop:
    bsf      EECON1, RD               ; B3 Read the data
    btfsc    EECON1, RD
```

```
    goto     $-1                          ; wait for read to
complete
    banksel EEDATA                  ; Bx -> B2
    movf    EEDATA, w               ; B2

    banksel EEADR
    incf     EEADR, f                  ; B2 increment EEPROM
data pointer
    btfsc   STATUS, Z               ; B2 did we overflow?
    incf    EEADRH, f                ; B2 yes, increment high
byte of EEPROM data pointer

    call    SendEscapeByte          ; Bx -> B1
    call    AddCrcB1                ; B1

    BXtoBO                          ; Bx -> BO
    decfsz  DATA_COUNTL, F          ; BO
    goto     ReadEeprom                ; Bx Not finished then
repeat
    goto    SendChecksum            ; Bx

;                                                          In:
<STX>[<OxO6><ADDRL><ADDRH><OxOO><OxOO><BYTESL><BYTESH><DATA>.
..]<CRCL><CRCH><ETX>
; Out:  <STX>[<OxO6>]<CRCL><CRCH><ETX>
WriteEeprom:                        ; Bx
    incf     FSR, f                      ; Bx increment data
buffer pointer
    movf     INDF, w                     ; Bx read data from
buffer
    banksel EEDATA                  ; Bx -> B2
    movwf    EEDATA                     ; B2 load the least
significant byte holding register
    B2toB3                          ; B2 -> B3
    movlw   b'OOOOO1OO'              ; Setup for EEPROM data
writes
    movwf   EECON1                  ; B3
    call    StartWriteB3            ; B3
```

```
    btfsc    EECON1, WR                ; B3 wait for write to
complete before moving to next address
    goto    $-1

    banksel EEADR
    incf     EEADR, F                  ; B2/B3 Adjust EEDATA
pointer

    B2toB0                             ; B2 -> B0
    decfsz  DATA_COUNTL, f             ; B0
    goto    WriteEeprom                ; Bx
    goto    SendAcknowledge            ; B0

; ********************************************
; Send an acknowledgement packet back
;
; <STX><COMMAND><CRCL><CRCH><ETX>

; Some devices only have config words as FLASH memory. Some
devices don't have EEPROM.
; For these devices, we can save code by jumping directly to
sending back an
; acknowledgement packet if the PC application erroneously
requests them.
#ifndef FREE
EraseFlash:                                ; only PIC16F88/87
supports explicit FLASH erase commands
#endif
SendAcknowledge:                   ; B0
    movf    COMMAND, w             ; B0
    call     SendEscapeByte          ; B0/B1 -> B1 Send only
the command byte (acknowledge packet)
    call    AddCrcB1               ; B1

SendChecksum:                      ; Bx
    call    SendCRCWord            ; Bx -> B1

SendETX:                           ; Bx -> B1
    BXtoB0                         ; Bx -> B0
```

```
    movlw    ETX                          ; send end of text
condition
    call     SendHostByte                 ; BO/B1 -> B1
    goto     WaitForHostCommand           ; BO/B1

;
************************************************************
***************

;
************************************************************
***************
SendCRCWord:                              ; Bx -> B1
    banksel CRCL
    movf     CRCL, w                      ; B1
    call     SendEscapeByte               ; Bx -> B1
    movf     CRCH, w                      ; B1
    ;; fall through to SendEscapeByte routine below

; Write a byte to the serial port while escaping control
characters with a DLE
; first.
SendEscapeByte:                           ; Bx -> B1
    banksel TXDATA
    movwf    TXDATA                       ; B1 Save the data

    xorlw    STX                          ; Check for a STX
    bz       WrDLE                        ; No, continue WrNext

    movf     TXDATA, W                    ; B1
    xorlw    ETX                          ; Check for a ETX
    bz       WrDLE                        ; No, continue WrNext

    movf     TXDATA, W                    ; B1
    xorlw    DLE                          ; Check for a DLE
    bnz      WrNext                       ; No, continue WrNext

WrDLE:
```

```
    movlw   DLE                     ; Yes, send DLE first
    call    SendHostByte            ; BO/B1 -> B1

WrNext:
    movf    TXDATA, W               ; B1 Then send STX

SendHostByte:                       ; BO/B1 -> B1
    B1toBO                          ; B1 -> BO
    clrwdt
    btfss   PIR1, TXIF               ; BO Write only if TXREG
is ready
    goto    $-1

    banksel TXREG
    movwf   TXREG                   ; BO Start sending
    BOtoB1                          ; BO -> B1
    return
;
*************************************************************
***************

ReadHostByte:                       ; Bx -> BO
    BXtoBO                          ; Bx -> BO
    clrwdt
    btfss   PIR1, RCIF                ; BO Wait for data from
RS232
    goto    $-1

#ifdef BSR
    banksel RCREG
#endif
    movf    RCREG, W                ; BO Save the data
#ifdef BSR
    movlb   .0
#endif
    movwf   INDF                    ; Bx
    return

#ifdef USE_SOFTBOOTWP
```

```
    goto     BootloaderStart           ; this code -should-
never be executed, but in case of errant
    goto     BootloaderStart           ; execution or firmware
bugs, may protect against accidental writes.
#endif

StartWrite:                            ; B2/B3
    B2toB3                             ; B2 -> B3
StartWriteB3:
    clrwdt                             ; Bx
    movlw   0x55                       ; B3 Unlock
    movwf   EECON2                     ; B3
    movlw   0xAA                       ; B3
    movwf   EECON2                     ; B3
    bsf     EECON1, WR                 ; B3 Start the write
    nop                                ; Bx
    nop                                ; Bx
    return                             ; B3

    END
```